دليل المبرمج إلى OpenGL

إعداد المهندس

إياد هلالي



للطباعة والنشر والتوزيع

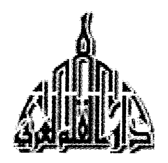
حلب – سورية

هاتف ٢٢١٥١١٢ _ ٢٢١٥٥٨٠

فاكس: ٢٢٥٤٩٢٨ _ ص.ب: ١٢٦٥٣

E-mail: bouraque@scs-net.org

www.bouraque.com



للطباعة والنشر والتوزيع

حلب – سورية

هاتف ٢١١٣١٢٩_ ٢١٢٢٥٩٩

فاكس: ٢٢١٢٣٦١ _ ص.ب: ٧٨

E-mail: qalamrab@scs-net

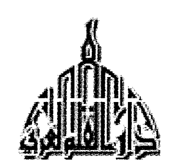
www.qalamarabi.net

الطبعة الأولى

١٤٢٥ ـ ٢٠٠٤



للطباعة والنشر والتوزيع
حلب ـ سورية
هاتف ٢٢١٥١١٢ ـ ٢٢١٥٥٨٠
فاكس: ٢٢٥٤٩٢٨ ـ ص.ب: ١٢٦٥٣
E-mail: bouraque@scs-net.org
www.bouraque.com



للطباعة والنشر والتوزيع
حلب ـ سورية
هاتف ٢١١٣١٢٩ـ ٢١٢٢٥٩٩
فاكس: ٢٢١٢٣٦١ ـ ص.ب: ٧٨
E-mail: qalamrab@scs-net
www.qalamarabi.net

مقـــدمة

الحمد لله نحمده ونستعينه ونستغفره ونتوب إليه و الصلاة و السلام على سيدنا محمد وعلى آله وصحبه .

يشهد الوقت الحالي تطوراً كبيراً في مجال البرمجيات،وخصوصاً في مجال برامج الرسوميات ثلاثية الأبعاد. مكتبة الرسوميات *OpenGL* عبارة عن واجهة برمجية لتجهيزات الرسوميات. يمكنك بواسطتها و بالاعتماد على إحدى لغات البرمجة (مثل *Visual C++*) بناء برامج رسومية ثلاثية الأبعاد. بقي أن نذكر أخيراً أن هناك الكثير من البرامج الشهيرة التي تعتمد في بناء رسومياتها على *OpenGL* و من أشهر هذه البرامج *3ds max*.

ماذا يتضمن هذا الكتاب؟

يعمل هذا الكتاب على شرح *OpenGL* بشكل مبسط وواضح و عملي من خلال تسعة فصول تتدرج في مستواها بحيث تمكن القارئ من فهم *OpenGL* و العمل معها بشكل جيد. تم وضع تطبيقات عملية في نهاية كل فصل لشرح ما ورد في ذلك الفصل من التوابع. ووضع في نهاية الكتاب أربعة ملاحق تشرح معظم توابع المكتبات التابعة لـــ *OpenGL* .

- يركز الفصل الأول على فهم *OpenGL* و طريقة عملها و صيغة أوامرها و المكتبات المرتبطة بها و كيفية بناء برنامج *OpenGL*.
- يعلمك الفصل الثاني المحاور و المستويات الإحداثية و رسم العناصر الهندسية.
- يشرح الفصل الثالث مبدأ عمل الكاميرا والتحويلات المرافقة لـــ*OpenGL* .
- يعلمك الفصل الرابع لوائح الإظهار و مدى أهميتها و كيفية استخدامها.

- يختص الفصل الخامس بشرح الألوان و أنماطها و التظليل في *OpenGL*.
- يشرح الفصل السادس الإضاءة و ما تضفيه من واقعية على عناصر المشهد.
- يشرح الفصل السابع مفهوم المزج في حالة توضع العناصر الشفافة أمام بعضها البعض و مفهوم صقل العناصر لإزالة التشويه و مفهوم الضباب.
- يعلمك الفصل الثامن كيفية رسم الخطوط **Fonts** في *OpenGL* و كيفية معالجة الصور *Images*.
- يشرح الفصل التاسع و الأخير عملية إكساء عناصر المشهد بالخرائط و المواد بحيث تصبح العناصر حقيقية.
- يشرح الملحق " أ " أوامر المكتبة الرئيسية لـــ *OpenGL* ووسائط كل أمر.
- يشرح الملحق " ب " أوامر المكتبة *GLU* ووسائط كل أمر.
- يشرح الملحق " ج " أوامر المكتبة *GLU* ووسائط كل أمر.
- يشرح الملحق " د " أوامر المكتبة *AUX* ووسائط كل أمر مع ذكر أمثلة عنها.

هذا و قد أرفقنا مع الكتاب قرصاً ليزرياً لتسهيل العمل على القارئ يحوي المجلدات:

- *Applications*: يحوي جميع النصوص البرمجية للتطبيقات الواردة في الكتاب.
- *Examples*:يحوي جميع النصوص البرمجية للأمثلة العملية الواردة في الكتاب.
- *Files*: يحوي ملفات المكتبات التابعة لـــ *OpenGL*. تستطيع نسخ هذه الملفات ووضعها في المكان المناسب حسب ما هو مشروح في الفصل الأول. و بدون هذه الملفات لا تعمل برامج *OpenGL*.
- *Pictures*: تحوي العديد من الصور المصنوعة بـــ *OpenGL*. يمكنك الاستفادة منها من هذه الصور لمعرفة إمكانيات *OpenGL*.
- *Projects* : يحوي هذا المجلد نصوصاً برمجيةً لمشاريع إضافية يمكنك الاستفادة منها لتعلم المزيد .

- *Tutorials* : يحوي العديد من الدروس العملية لبعض برامج *OpenGL* مع شرح مفصل باللغة الإنكليزية لكل درس .

أرجو من الله تعالى أن ينفع به القارئ وأن ينفعنا بع وأن يكون خالصاً لوجهه الكريم ، وأن يوفقنا للخير . وآخر دعوانا أن الحمد لله رب العالمين .

شكر

أتوجه بالشكر إلى كل الذين ساهموا في إنجاز هذا الكتاب ، وأخص بالشكر الزميلات المهندسات في مخبر البرمجة في كلية هندسة الحاسبات التالية أسماؤهن: ديمة أسود ، لينا مكن ، هلا الحسن ، لما علي ، صفاء هنو ، سوسن اسجيع ، حلا قشقش. لما قدموه من جهود كبيرة ساعدت في إنجاز هذا الكتاب.

المهندس : إياد هلالي

الفصل الأول

مقدمة إلى OpenGL

ستتعلم في هذا الفصل المواضيع التالية:

- ➢ ما هي *OpenGL* ؟
- ➢ لماذا *OpenGL* ؟
- ➢ من يتحكم بـ *OpenGL* ؟
- ➢ كيف تعمل *OpenGL* ؟
- ➢ صيغة أوامر *OpenGL*
- ➢ *OpenGL* كآلة حالة
- ➢ المكتبات المرتبطة بـ *OpenGL*
- ➢ برنامج إطار *OpenGL*
- ➢ تطبيقات عملية

ما هي OpenGL ؟

تعني OpenGL (*Open Graphic Library*) مكتبة الرسوميات المفتوحة وهي واجهة برمجية للتجهيزات *hardware* الخاصة بالرسوميات . تتألف هذه الواجهة من عدة مئات من التوابع التي تسمح لك ولمبرمجي الرسوميات بتحديد العناصر والعمليات المطلوبة لانتاج صور رسومية ملونة عالية الجودة ثلاثية الأبعاد . الاختلاف بين الكثير من هذه التوابع بسيط ، لذلك لدينا فعلياً فقط ١٢٠ تابع (أمر) أساسي .

يمكن أن تعمل برامج *OpenGL* عبر الشبكات حتى لو كان الحاسب العميل والمخدم مختلفين من ناحية التجهيزات .

تتميز OpenGL بالمزايا التالية

- تقدم OpenGL تسريعاً ثلاثي الأبعاد على مستوى التجهيزات .
- تعالج التطبيقات و الألعاب في أيامنا هذه كمية ضخمة من البيانات في الزمن الحقيقي باستخدام العناصر الهندسية والإضاءة في الزمن الحقيقي وعمليات الاقتطاع والتحويلات والإكساء التابعة لـــ OpenGL.
- تجعل OpenGL التأثيرات ثلاثية الأبعاد التي تحدث في الزمن الحقيقي محتملة.
- يضيف تسريع OpenGL على مستوى التجهيزات تفاصيلاً و تأثيرات خاصة إلى الصور دون التأثير على الأداء. وكمثال على ذلك إضافة ضباب في الزمن الحقيقي و صقل و ظلال و غباشة متحركة و شفافية و تراكيب ثلاثية الأبعاد.
- صُممت OpenGL لدعم ابتكارات مستقبلية في مجال البرمجيات والتجهيزات .

- تستطيع التقنيات الممتدة لــ OpenGL التعامل الأساسي مع المزايا البرمجية و التجهيزات التي لم تكن موجودة عند إنشاء OpenGL. لذلك يجب عليك أن تكون واثقاً من أنك ستحصل على أداء أمثلي لتطبيقاتك و ألعابك مع تطور تكنولوجيا التجهيزات.
- تنفذ OpenGL على أنظمة تشغيل مختلفة
- تستطيع و بسهولة نقل تطبيقات و ألعاب تدعم OpenGL

من نظام تشغيل لآخر. و هذا يجعل تطبيقات *OpenGL* قابلة للحمل، أي تعمل على أنظمة تشغيل مختلفة مثل *Windows* و *Linux* و *OS/2*.

لماذا *OpenGL* ؟

تعتبر الإمكانيات الرسومية وخصوصاً الرسوميات ثلاثية الأبعاد من الأشياء الضرورية لخطط *Microsoft Windows* ، وقد عملت *Microsoft* مع *SGI* لإنشاء مكتبة رسومية ثلاثية الأبعاد *3D* (*OpenGL*) وتتمثل مصلحة *Microsoft* من هذه المكتبة في جعلها منافس خطير في سوق محطات العمل ، أما مصلحة *SGI* فتتمثل في الحصول على فرصة لدخول سوق الحواسيب الشخصية . قدمت شركة *Microsoft* المكتبة *Render Morphics* عام ١٩٩٥. وهي عبارة عن مكتبة واجهة تطبيقات *API* (*Application Program Interface*) مصممة للألعاب (مكتبة غرضية التوجه سريعة وقوية) وتشكل مجموعة رئيسية لــ *OpenGL* . فكلاهما يمتلك محرك تصيير يعتمد النقاط *vertices*.

من يتحكم بـ *OpenGL* ؟

صممت بنية *OpenGL* من قبل *ARB* (*Architecture Review Board* *OpenGL*) وهي عبارة عن مجموعة من الشركات المهتمة بتأمين مشاهد *3D* قياسية عبر تجهيزات متنوعة وأنظمة تشغيل مختلفة .

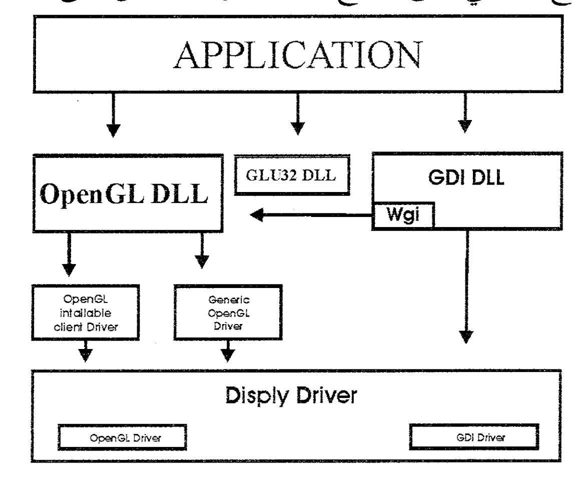
عملت شركتان من ARB وهما Microsoft و SGI (Silicon Graphics Incorporated) مع بعضها البعض على OpenGL منذ عام ١٩٩١ وقد حفظت OpenGL القياسية لـــ ARB .

تنفذ مراجعات دورية لـــ OpenGL من قبل جماعة مستخدمي OpenGL كما تعدل مزايا OpenGL وفحوص الملائمة لها كلما ظهرت نسخة جديدة من OpenGL.

تستخدم فحوص الملائمة لحصول منتج ما على حقوق OpenGL ولكي يقال عن منتج أنه خاضع لـــ OpenGL ، يجب أن ينجح في فحوص الملائمة وتتحقق هذه الفحوص عندما يكون المنتج يدعم جميع مزايا OpenGL.

كيف تعمل OpenGL ؟

تعمل OpenGL بطريقة مشابهة لعمل واجهة الرسوميات GDI (Graphical Device Interface) فهي تمتلك فقط طبقة اضافية تستطيع البرامج عنونتها . تكمن وظائف GDI في مكتبة الربط الديناميكية GDI32.DLL التي تحّمل كلما استدعى برنامج وظيفة GDI (تستخدم لاظهار الصناديق الحوارية والأزرار والصور والعناصر الرسومية) . كلما نفذ برنامج استدعاء OpenGL ، تحّمل عند ذلك مكتبات الربط الديناميكية GLU32.DLL و OpenGL32.DLL . يظهر الشكل التالي كيف تنفذ الاستدعاءات في برنامج تطبيقي حتى تصبح نقاطاً ضوئية تعرض على الشاشة .



تتألف واجهة التطبيقات API (Application Program Interface) التابعة لـ *OpenGL* في ويندوز من حوالي ١٥٠ وظيفة (تابع) . تقدم توابع إضافية لـ *OpenGL* مع التعاريف التي تأتي مع بطاقات الإظهار .

صيغة أوامر OpenGL

تستخدم *OpenGL* بادئة تعبر عن المكتبة المأخوذ منها الأمر (مثلاً تعبر البادئة *gl* عن الأوامر المأخوذة من المكتبة *opengl32.lib*) ، كما تستخدم حروف كبيرة لبداية كل كلمة تؤلف اسم الأمر مثل *glClearColor* .

تبدأ الثوابت المعرفة ضمن *OpenGL* بالسابقة *GL* وتستخدم حروفاً كبيرة ورمز الشرطة السفلية "_" *Underscore* لتفصل الكلمات عن بعضها مثل *GL_COLOR_BUFFER_BIT* .

تضاف أحياناً حروف إلى أسماء الأوامر مثل *3f* في الأمر *glColor3f()* ، حيث يدل الرقم *3* على وجود ثلاثة وسائط للأمر أما الحرف *f* فيدل على أن الوسائط من نوع أعداد فاصلة عائمة . بعض أوامر *OpenGL* تقبل حتى ٨ وسائط مختلفة .

يبين الجدول التالي الحروف المستخدمة لتحديد أنواع بيانات الوسائط :

الحرف	نوع البيانات	النوع المقابل بلغة c	تعريف النوع بـ OpenGL
b	عدد صحيح 8 Bit	Signed char	GLbyte
s	عدد صحيح 16 Bit	short	GLshort
i	عدد صحيح 32 Bit	Long	GLint , GLsizei
f	فاصلة عائمة 32 Bit	Float	GLfloat,GLclampf
d	فاصلة عائمة 64 Bit	double	GLdouble , GLclamped
ub	عدد صحيح غير مؤشر 8 Bit	Unsigned char	GLubyte , GLboolean
us	عدد صحيح غير مؤشر 16 Bit	Unsigned short	GLushort
ui	عدد صحيح غير مؤشر 32 Bit	Unsigned long	GLunit , GLenum , GLbitfield



مثـــــــال

الأمران*glVertex2i(1,3)* و *glVertex2f(1.0,3.0)* متكافآن عدا أن الأول يحدد إحداثيات النقطة كأعداد صحيحة بطول *32 bit* أما الثاني فيحددها كأرقام فاصلة عائمة أحادية الدقة . يمكن أن تأخذ بعض أوامر *OpenGL* حرفاً أخيراً (*v*) يشير إلى أن الأمر يشير إلى شعاع أو مصفوفة من القيم بدلاً من سلسلة وسائط مستقلة . أغلب الأوامر يمكن استعمالها مع شعاع *v* أو بدون شعاع .

المثال التالي هو أمر لتحديد اللون تمت كتابته مع شعاع وبدون شعاع

بدون شعاع

```
glColor3f(1.0,0.0,0.0);
```

مع شعاع

```
float color_ array[ ] ={1.0,0.0,0.0};
glColor3fv( color-array);
```

تعرف *OpenGL*الثابت *GLvoid* بدلاً من*void* إذا كنت مبرمجاً بلغة *c*

كآلة حالة *OpenGL*

توضع *OpenGL* في حالات مختلفة تبقى متأثرة بها حتى نقوم بتغييرها . مثلاً اللون الحالي متحول حالة يحتفظ بالقيمة التي نسندها إليه (مثلاً أبيض أو أحمر ...) بحيث ترسم كل الكائنات بعد هذه التعليمة باستخدام هذا اللون إلى أن نعطي هذا المتحول لوناً آخر .

تستخدم متحولات الحالة للتحكم بالمظهر الحالي والنماذج المنقطة والخطوط والمضلعات وأنماط رسم المضلعات وخصائص الإضاءة .

تشير معظم متحولات الحالة إلى أنماط يمكن تفعيلها أو تعطيلها من خلال الأمرين *glEnable( )* و *glDisable ()* .

يمتلك كل متحول حالة قيمة افتراضية ، ويمكن من وقت لآخر الاستعلام من النظام عن القيمة الحالية للمتحول بواسطة أحد الأوامر الأربعة التالية :

- glGetBooleanv( )
- glGetDoublev( )

- glGetFloatv( )
- glGetIntegerv( )

اختيار الأمر المناسب حسب نوع معطيات الإجابة التي سينتجها الأمر.

يمكن أيضاً تخزين واستعادة قيم من مجموعة متحولات الحالة بواسطة مكدس خصائص باستخدام الأمرين :

- glPushAttrib( )
- glPopAttrib ( )

المكتبات المرتبطة بـ *OpenGL*

تحوي *OpenGL* مجموعة قوية ولكن أولية من أوامر التصيير ، وكل الرسوم ذات المستوى الأعلى يجب أن تنفذ باستخدام هذه الأوامر .

قد نرغب أحياناً بكتابة مكتبة خاصة تُحمل على *OpenGL* لتسهيل المهام البرمجية و لتسهيل تعامل *OpenGL* مع نظام الإطارات . نورد أشهر هذه المكتبات :

١. ***GL***

تعتبر من أشهر مكتبات *OpenGL* و أكثرها استخداماً.

سنتعامل من خلال هذا الكتاب مع معظم أوامر هذه المكتبة. تستطيع أيضاً الحصول على شرح لجميع أوامر هذه المكتبة من خلال الملحق " أ ". تستخدم أوامر هذه المكتبة لإنشاء العناصر الهندسية وتطبيق التحويلات والإضاءة والإكساء بالخرائط...الخ.

تبدأ جميع أوامر (توابع) هذه المكتبة بالبادئة *gl*. الملفات المسؤولة عن هذه المكتبة ضمن لغة البرمجة *Visual C++ 6.0* هي:

- الملف OpenGL32.dll

يوضع ضمن المجلد *System* لنظام التشغيل *Windows*.

- الملف OpenGL32.lib

يتوضع ضمن المجلد *lib* التابع للغة البرمجة السابقة و يأتي هذا البرنامج مع تلك اللغة و لا حاجة لإضافته إليها من مصدر خارجي.

- الملف GL.h

يتوضع ضمن المجلد *GL* الموجود ضمن المجلد *include* التابع للغة البرمجة السابقة و يأتي هذا البرنامج أيضاً مع اللغة.

٢. ***GLU (OpenGL Utility Library)***

مكتبة خدمات *OpenGL* تتضمن مهام مختلفة مثل رسم كرة و أسطوانة و منحني و تغيير حجم صورة ...الخ.

تبدأ جميع أوامر هذه المكتبة بالبادئة *glu*. و ملفات هذه المكتبة:

- Glu32.dll

يوضع ضمن المجلد *System* لنظام التشغيل *Windows*.

- Glu32.lib

يرفق مع لغة *VC++* و يتوضع ضمن المجلد *Lib*.

- Glu.h

يرفق مع لغة *VC++* و يتوضع ضمن المجلد *include\gl*.

يلخص الملحق " ب " أوامر هذه الكتبة.

٣. ***GLUT (OpenGL Utility Toolkit)***

عبارة عن مجموعة أدوات خدمية تابعة لـــ *OpenGL* مكتوبة بواسطة مارك كيلغارد *Mark Kilgard*. استخدام هذه المكتبة سهل و مفيد، و سنستخدم هذه المكتبة لتهيئة و إنشاء إطار *OpenGL*. تستطيع الذهاب للملحق " ج " لمشاهدة معظم أوامر هذه المكتبة.

تبدأ جميع أوامر هذه المكتبة بالبادئة *glut*. الملفات المسؤولة عن هذه المكتبة:

- glut32.dll

يتم وضعه ضمن المجلد *System* التابع لنظام التشغيل *Windows*.

- Glut32.dll

ضعه ضمن المجلد *lib* للغة البرمجة *VC++ 6.0*.

- Glut.h

ضعه ضمن المجلد *include\gl*.

٤. AUX *(Auxiliary Library)*

تستطيع بواسطة هذه المكتبة رسم العديد من العناصر الهندسية ثلاثية الأبعاد وذلك إما بطريقة سلكية *wire* أو بطريقة مصمتة *Solid* . تستطيع بواسطة هذه المكتبة رسم كرة ومكعب وأسطوانة ومخروط وعناصر أخرى . يشرح الملحق " د " معظم أوامر هذه المكتبة . تبدأ أوامر هذه المكتبة بالبادئة *aux* . الملفات المسؤولة هن هذه المكتبة :

- GlAux.lib

  يرفق مع لغة *VC++* ويتوضع ضمن المجلد *Lib* .

- GlAux.h

  يرفق مع لغة *VC++* ويتوضع ضمن المجلد *include\gl* .

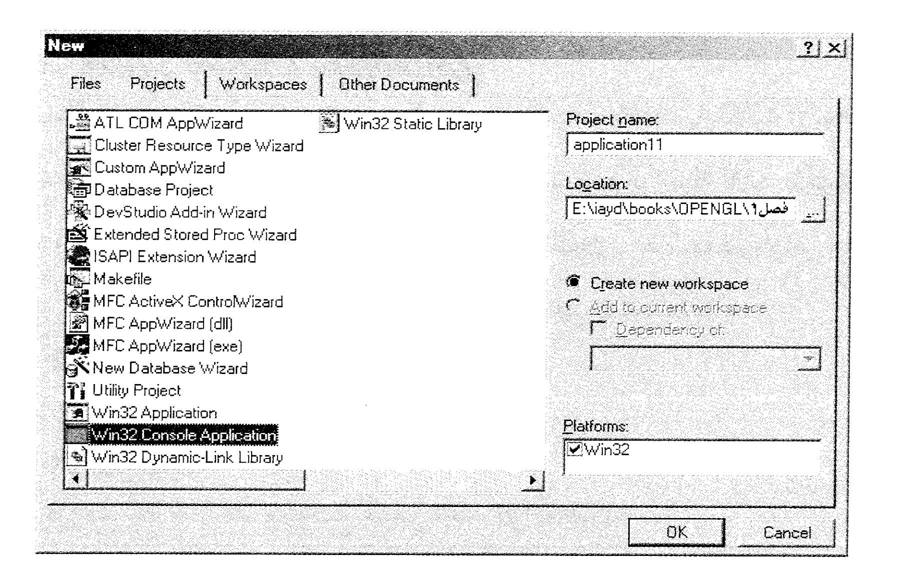
ملاحظة 	الملفات السابقة ضرورية لتنفيذ برامج OpenGL . يأتي بعض هذه الملفات مع لغة البرمجة VC++ 6.0 و بعضها يتم تحميله من الانترنت. و لتوفير الوقت و العناء فقد وضعنا جميع هذه الملفات في المجلد Files المتوضع ضمن القرص الليزري المرفق مع هذا الكتاب. تستطيع نسخ تلك الملفات و وضعها في المكان المخصص لها كما هو مبين في الفقرة السابقة

إعداد لغة البرمجة *Visual c++* لتنفيذ برامج *OpenGL*

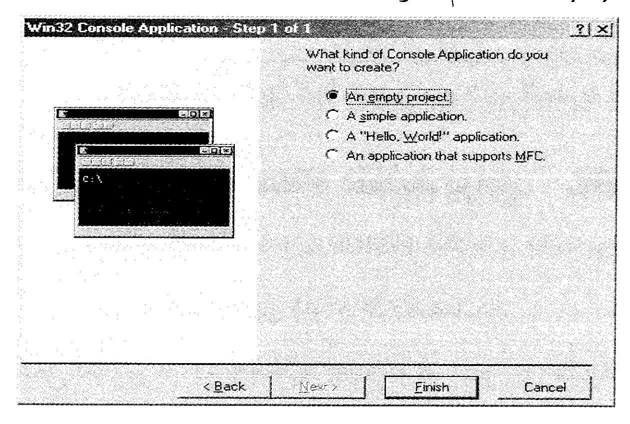
اتبع الخطوات التالية لإعداد *Visual C++* لكتابة و تنفيذ برامج *OpenGL*:

١. انسخ الملفات السابقة إلى الموقع المخصصة لها.

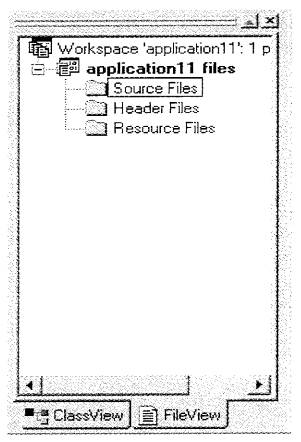
٢. شغل لغة البرمجة *Visual C++6.0* ثم انتقل إلى القائمة *File* و انتق منها الأمر *New*. يظهر عند ذلك مربع الحوار *New*. حدد ضمن علامة التبويب *Projects* نوع المشروع *Win32 Console Application* ثم اكتب اسم المشروع (مثلاً *application11*)ضمن حقل*Project Name* و حدد مكان حفظ المشروع ضمن *Location* كما هو مبين في الشكل التالي:



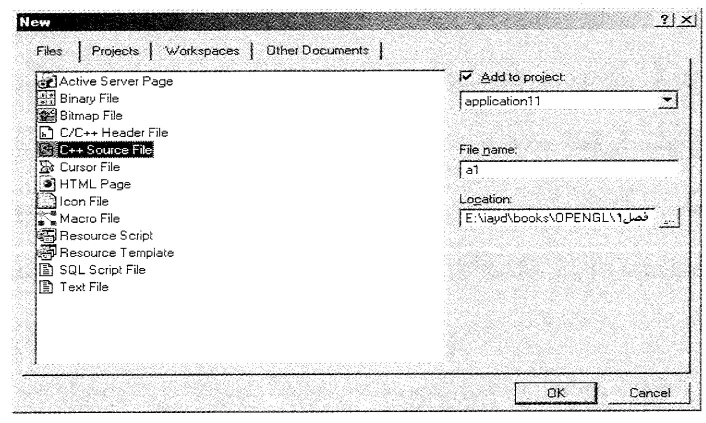
انقر بعد ذلك الزر *OK*. يظهر مربع حوار يطلب منك تحديد نوع المشروع. اختر مشروعاً فارغاً *An empty Project* ثم انقر *Finish*.



يظهر بعد ذلك مربع حوار تأكيد الإنشاء، انقر *OK*. انتقل بعد ذلك إلى شريط الأدوات *Workspace* الموجود في القسم اليساري من إطار *VC++* الرئيسي وانقر فوق علامة التبويب *FileView* وحدد منها المجلد *Source Files* كما هو مبين بالشكل التالي :



سنضيف الآن ملف *C++* إلى *Source Files* . انتقل إلى القائمة *File* واختر منها *New* ثم انتقل إلى علامة التبويب *Files* وحدد *C++ Source File* واكتب اسماً للملف ضمن حقل *File name* (مثلاً *a1*) ثم انقر الزر *Ok* .

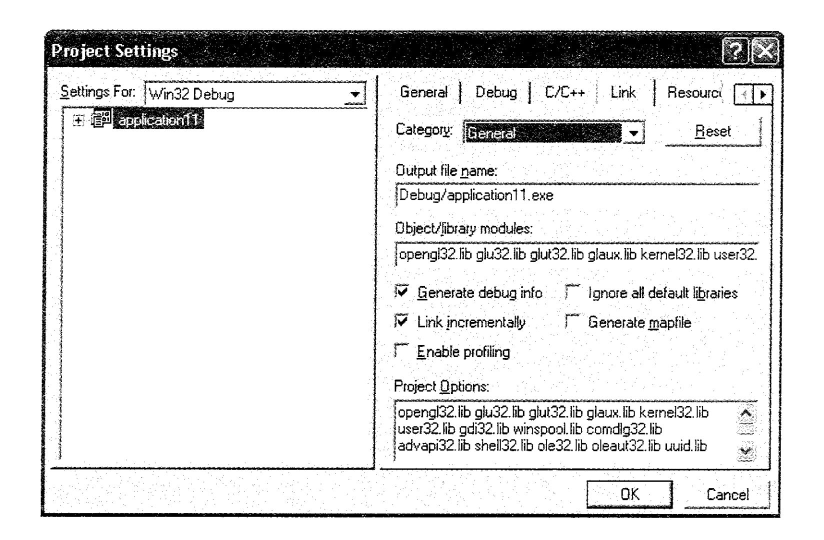


سنكتب النصوص البرمجية لتطبيقاتنا ضمن هذا الملف .

٣. اربط مكتبات *OpenGL* و ذلك بالانتقال إلى القائمة *Project* ثم انتقاء الخيار *Settings*. يظهر عند ذلك مربع الحوار *Project Settings*. انتقل إلى علامة التبويب *Link* واكتب ضمن حقل *Object/library modules* و في بداية السطر (قبل *Kernal32.lib*) أسماء المكتبات كما يلي:

Opengl32.lib glu32.lib glut32.lib glaux.lib

يبين الشكل التالي ذلك:



انقر بعد ذلك الزر *ok* . لغة البرمجة الآن جاهزة لكتابة وتنفيذ برامج *OpenGL*.

برنامج إطار *OpenGL*

حتى تتمكن من تنفيذ أوامر *OpenGL* و إظهار الرسوميات، أنت بحاجة إلى إطار لإظهار تلك الرسوميات (ناتج تنفيذ الأوامر). تشرح هذه الفقرة بشكل مفصل برنامج يستخدم لتجهيز إطار *OpenGL*، و سنستفيد من هذا البرنامج لعرض رسوميات *OpenGL* ضمنه و تأثيراتها الأخرى (كالإضاءة و الإكساء ...). سنستخدم المكتبة *Glut* لإنشاء برنامج الإطار هذا. و لنبدأ بشرح هذا البرنامج:

يُصرح في البداية عن ملفات العناوين *headers* للمكتبات التي سنستخدمها كما يلي:

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
```

نصرح بعد ذلك عن التابع *redraw* الذي سنستخدمه لرسم العناصر في *OpenGL*، حيث سنضع جميع برامج و أمثلة *OpenGL* ضمن هذا التابع.

```
static void redraw(void);
```

نبدأ بعد ذلك بالتابع الرئيسي في لغة *C++* و هو *main*.

```
int main(int argc, char **argv)
{
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowPosition(100,100);
  glutInitWindowSize(400,400);
  glutCreateWindow("My first GLUT program");
  glutDisplayFunc(redraw);
```

يحوي المتحول *argc* السابق عدد الوسائط المحررة لبرنامجنا، في حين يمثل *argv* مؤشراً إلى *argc*. لا تقلق بشأن هذه المؤشرات فهي تمرر إلى التابع *glutInit* الذي يستخدم لتهيئة إطار *Glut*.

يستخدم التابع *glutInitDisplayMode( )* لإعداد نمط الإظهار. سنستدعي هذا التابع وفق الوسائط التالية :

GLUT_RGB: يستخدم لإضافة ذاكرة مؤقتة *buffer* لألوان *RGB* ضمن الإطار الخاص بنا.

GLUT_DOUBLE: يستخدم لإضافة ذاكرة مؤقتة مزدوجة *Double*.

تمكننا الذاكرة المؤقتة المزدوجة من إنهاء الرسم قبل إرساله إلى الشاشة تجنباً لحدوث الوميض.

GLUT_DEPTH: يضيف ذاكرة مؤقتة للعمق إلى إطارنا. تستخدم هذه الذاكرة المؤقتة لتحديد بعد العناصر عن الكاميرا (عين الناظر).

يستخدم التابع *glutInitWindowPosition()* لتحديد موقع الإطار (احداثيات *x,y* للإطار) ، أما التابع *glutInitWindowSize()* فيستخدم لتعيين أبعاد الإطار (العرض والارتفاع) .

ينشئ التابع *glutCreateWindow ( )* الإطار الذي سنرسم ضمنه العناصر. يحدد *glutDisplayFunc ( )* التابع المستخدم للرسم (التابع *redraw*). سنطبق بعد ذلك

مصفوفة إسقاط *Projection matrix* ، تستخدم لتحديد موقع الكاميرا بالنسبة للشاشة. سنتحدث عن هذا التحويل في الفصل ٣ بشكل مفصل.

يبين النص البرمجي التالي هذه المصفوفة :

```
glMatrixMode(GL_PROJECTION);
gluPerspective(45,1.0,10.0,200.0);
```

سنستبدل الآن المصفوفة السابقة بمصفوفة التحويل *modelView*. و هي عبارة عن مصفوفة 4×4 تحول النقاط المرسومة من إحداثياتها الحقيقية (الفضاء الحقيقي) إلى إحداثيات تتعلق بالكاميرا (سنتحدث عنها في الفصل ٣). يستخدم التابع

glutMainLoop( ) لتكرار إظهار إطار *Glut* بشكل مستمر.

```
glMatrixMode(GL_MODELVIEW);
glutMainLoop();
return 0;
}
```

نكتب بعد ذلك التابع *redraw( )* الخاص بالرسم و نضع فيه العناصر التي سنرسمها.

```
static void redraw(void)
{
```

توضع تعليمات العناصر المراد رسمها هنا

```
}
```

و بذلك يصبح برنامج الإطار الكامل كما يلي :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>

static void redraw(void);
int main(int argc, char **argv);

int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB| GLUT_DOUBLE | GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
```

```
  glutCreateWindow("My first GLUT program");
  glutDisplayFunc(redraw);
  glMatrixMode(GL_PROJECTION);
  gluPerspective(45,1.0,10.0,200.0);
  glMatrixMode(GL_MODELVIEW);
  glutMainLoop();
  return 0;
}
static void redraw(void)
{
}
```

ملاحظة 	يجب الانتباه لحالة الأحرف (كبيرة أو صغيرة) عند كتابة الأوامر في لغة VC++، لأن هذه اللغة حساسة لحالة الأحرف.

تطبيقات عملية

سنعمل الآن على كتابة بعض البرامج العملية و تنفيذها و مشاهدة النتيجة. و لكن قبل أن نبدأ بعرض التطبيقات سنشرح بشكل مختصر بعض أوامر *Open* المستخدمة في هذه التطبيقات، و سنعود لدراسة هذه الأوامر بشكل مفصل من خلال الفصول القادمة:

1) glClearColor(1.0,1.0,1.0,0.0);

تعيين لون المسح الحالي (لون خلفية أبيض) لاستخدامه في مسح الذاكرة المؤقتة للون (تستخدم الذاكرة المؤقتة للون لتخزين ألوان النقاط الضوئية) .

2) glClearDepth(1.0);

تعيين قيمة كل نقطة ضوئية في ذاكرة العمق المؤقتة (يمثل العمق قياس بعد النقطة الضوئية عن عين الناظر) فالقيمة الموجبة تعني اقتراب الرسم من عين الناظر أما القيمة السالبة فتعني ابتعاد الرسم باتجاه الشاشة .

3) glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

مسح الذواكر المؤقتة *buffers* الخاصة باللون والعمق حسب قيم المسح الحالية.

4) glLoadIdentity();

يستبدل المصفوفة الحالية بالمصفوفة الواحدية وهذا يعيدنا إلى مركز الشاشة *(0,0,0)* .

5) glTranslatef(-1.5f,0.0f,-6.0f);

يحركنا عبر الشاشة لتحديد مكان الرسم الجديد. الحركة لا تتم من مركز الشاشة وإنما من الموقع الحالي للرسم .

6) glColor3f(1.0, 1.0, 1.0);

يعين لون الرسم الحالي للعناصر . جميع العناصر الواردة بعد هذا الأمر سترسم بلونه.

7) glBegin(GL_TRIANGLES);

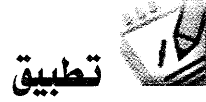
يعين بداية لائحة النقاط المرسومة ،ويحوي وسيط لتحديد نوع العنصر الهندسي المرسوم. مثلاً *GL_TRIANGLES* لرسم عنصر هندسي ثلاثي النقاط.

8) glEnd();

يحدد نهاية لائحة النقاط المرسومة.

9) glVertex2f(0.5, 0.5);

تحديد إحداثيات النقاط المراد رسمها.



تطبيق ١

برنامج *OpenGL* لإظهار مستطيل أسود على خلفية بيضاء

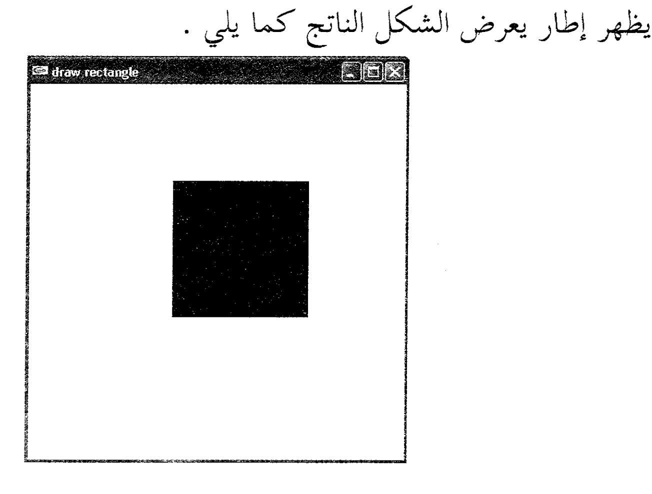
لتنفيذ هذا التطبيق ، أنشئ برنامج *OpenGL* فارغ اسمه *application11* كما تعلمت ذلك في فقرة " **إعداد لغة البرمجة *Visual c++* لتنفيذ برامج *OpenGL*** " ، ثم اكتب النص البرمجي التالي ضمن ملف *c++* (الملف *a1.cpp* في البرنامج الفارغ السابق) :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("Application11");
 glutDisplayFunc(redraw);
```

```
  glMatrixMode(GL_PROJECTION);
  gluPerspective(45,1.0,10.0,200.0);
  glMatrixMode(GL_MODELVIEW);
  glutMainLoop();
  return 0;
}
static void redraw(void)
{
  glClearColor(1.0,1.0,1.0,0.0);
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  glColor3f(0.0,0.0,0.0);
  glTranslatef(-1.5f,0.0f,-100.0f);
  glColor3f(1.0, 1.0, 1.0);
 glBegin(GL_POLYGON);
     glVertex2f(-30, -30);//الزاوية اليسارية السفلية للمستطيل
     glVertex2f(-30, 30);//الزاوية اليسارية العلوية للمستطيل
     glVertex2f(30, 30);//الزاوية اليمينية العلوية للمستطيل
     glVertex2f(30, -30);//الزاوية اليمينية السفلية للمستطيل
glEnd();
glutSwapBuffers();
}
```

نفذ بعد ذلك التطبيق بضغط زر التنفيذ ! من شريط الأدوات *Build MiniBar*

يظهر إطار يعرض الشكل الناتج كما يلي .



تستطيع مشاهدة التطبيق كاملاً بالانتقال إلى القرص المضغوط المرفق مع هذا الكتاب ، وفتح المجلد applications ثم الانتقال إلى المجلد application11 وتنفيذالملف application11.dsw، ثم تشغيل زر التنفيذ ضمن واجهة ++VC. سنضع جميع التطبيقات المشروحة في فصول هذا الكتاب ! ضمن هذا المجلد. تستطيع العودة إليها عند الضرورة .	ملاحظة 

تطبيق ٢

برنامج *OpenGL* لرسم مثلث ومربع

أنشئ تطبيقاً اسمه *application12* وأنشئ ضمنه الملف *a2.cpp* ثم اكتب النص البرمجي التالي ضمن الملف *a2.cpp* :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
 GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("Application12");
 glutDisplayFunc(redraw);
 glMatrixMode(GL_PROJECTION);
 gluPerspective(45,1.0,10.0,200.0);
 glMatrixMode(GL_MODELVIEW);
 glutMainLoop();
 return 0;
}
static void redraw(void)
```

```
{
 glClearColor(1.0,1.0,1.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glColor3f(0.0,0.0,0.0);
 glTranslatef(-20.0,0.0,-100.0);
 glBegin(GL_TRIANGLES);         // رسم المثلث

       glVertex3f( 0.0, 10.0, 0.0);       // النقطة العلوية

       glVertex3f(-10.0,-10.0, 0.0);      // النقطة السفلية اليسارية

       glVertex3f( 10.0,-10.0, 0.0);      // النقطة السفلية اليمينية

 glEnd();
 glTranslatef(40.0,0.0,0.0);
 glBegin(GL_QUADS);                        // رسم المربع

       glVertex3f(-10.0, 10.0, 0.0);      // الزاوية العلوية اليسارية

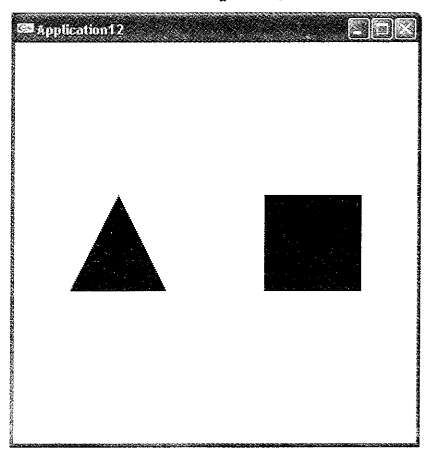
       glVertex3f( 10.0, 10.0, 0.0);      // الزاوية العلوية اليمينية

       glVertex3f( 10.0,-10.0, 0.0);      // الزاوية السفلية اليمينية

       glVertex3f(-10.0,-10.0, 0.0);      // الزاوية السفلية اليسارية

 glEnd();
 glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



الفصل الثاني

رسم العناصر الهندسية

ستتعلم في هذا الفصل المواضيع التالية:

- المحاور والمستويات الإحداثية
- مسح إطار ***OpenGL*** إلى لون معين
- النقاط (الرؤوس) والخطوط والمضلعات
- أساسيات الرسم الهندسي في ***OpenGL***
- الشكل العام للأمرين ***glBegin( )*** و ***glEnd( )***
- إظهار النقاط و الخطوط و المضلعات
- النواظم (الأشعة الاعتيادية) ***Normal Vectors***
- تطبيقات عملية

المحاور والمستويات الإحداثية

لدينا ثلاثة محاور هي *x,y,z* .

X : تكون الحركة يمين مركز الاحداثيات بقيم موجبة و يساراً بقيم سالبة .

Y : تكون الحركة أعلى مركز الاحداثيات بقيم موجبة و أسفل مركز الاحداثيات بقيم سالبة .

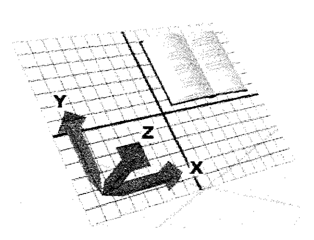
Z : تكون الحركة نحو الداخل (للشاشة) بقيم سالبة ونحو الخارج (للمستخدم) بقيم موجبة .

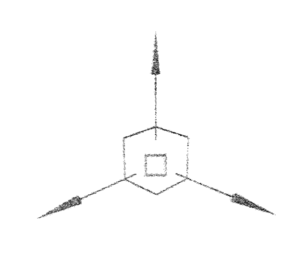
ولدينا ثلاثة مستويات :

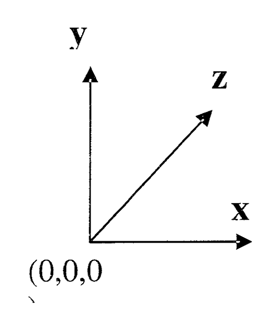
xy : الحركة وفق المحورين *xy*

yz : الحركة وفق المحورين *yz*

zx : الحركة وفق المحورين *zx*







مسح إطار *OpenGL* إلى لون معين

تعتبر هذه العملية إحدى الخطوات التحضيرية لبدء عملية الرسم . اللون الذي تستخدمه للون الخلفية يعتمد على التطبيق ، فمن أجل معالج النصوص يمكن أن تمسح الخلفية إلى لون أبيض أما إذا كنت ترسم منظراً للفضاء ، فعليك مسح الفراغ بلون أسود قبل البدء برسم النجوم والكواكب . سنقدم الآن مثالاً عن كيفية مسح إطار *OpenGL* بلون أسود .



مثـــــال

```
glClearColor(0.0,0.0,0.0,0.0);
glClear(GL_COLOR_BUFFER_BIT);
```

السطر الأول يجهز لون مسح الإطار باللون الأسود أما السطر الثاني فيمسح كامل الإطار حسب اللون الحالي الموجود في الأمر *glClearColor* .

يشير الوسيط الوحيد لـــ *glClear* إلى نوع الذاكرة المؤقتة *buffer* التي يجب مسحها. في مثالنا يتم فقط مسح الذاكرة المؤقتة للون بينما تبقى الصورة المعروضة على الشاشة .

يتم بشكل نموذجي تجهيز لون المسح مرة واحدة في بداية تطبيقك ، ثم تمسح الذواكر المؤقتة *buffers* كلما احتجت لذلك . يحفظ *OpenGL* مسار لون المسح الحالي كمتحول حالة بدلاً من أن يطلب منك تحديده في كل مرة يمسح فيها الذاكرة المؤقتة .



مثـــــال

سنمسح الآن ذاكرتي اللون والعمق المؤقتتين :

```
glClearColor(0.0,0.0,0.0,0.0);
glClearDepth(0.0);
glClear(GL_COLOR_BUFFER_BIT| GL_DEPTH_BUFFER_BIT);
```

السطر الأول يجهز لون المسح (الأسود) أما السطر الثاني *glClearDepth ( )* فيحدد القيمة التي يجب أن تعين لكل نقطة ضوئية في الذاكرة المؤقتة للعمق *Depth Buffer*.

يتضمن السطر الثالث الأمر *glClear* الذي تتألف وسائطه من الوسيط المنطقي *OR* (|) مع جميع الذواكر المؤقتة التي يجب مسحها .

يبين الجدول التالي الذواكر المؤقتة الممكن مسحها بأمر *glClear* .

الذاكرة المؤقتة	الاسم	الوصف
الذاكرة المؤقتة للون **Color buffer**	**GL_COLOR_BUFFER_BIT**	تخزين ألوان النقاط الضوئية**pixels** للخلفية
الذاكرة المؤقتة للعمق **Depth buffer**	**GL_DEPTH_BUFFER_BIT**	تدعى أحياناً **Z buffer** وتستخدم لتخزين قيم العمق لكل نقطة ضوئية والتي تمثل قياس المسافة عن عين الناظر (النقطة ضوئية ذات العمق الموجب تغطي النقطة ضوئية ذات العمق الموجب الأصغر منه أو ذات العمق السالب)
الذاكرة المؤقتة الحاجبة **Stencil buffer**	**GL_STENCIL_BUFFER_BIT**	تستخدم لقصر الرسم على مناطق محددة من الشاشة . وتشبه عملية وضع قالب ورقي عندما يراد بخ صورة معينة. لا ترسم العناصر الموضوعة في هذه الذاكرة المؤقتة ، فمثلاً عند رسم مشهد لصورة سيارة تتحرك عبر منظر ما ، يتم وضع السيارة وما تحويه ضمن الذاكرة المؤقتة هذه ويتم تغيير المشهد المحيط فقط .
الذاكرة المؤقتة التراكمية **Accumulation buffer**	**GL_ACCUM_BUFFER_BIT**	تستخدم لمراكمة مجموعة صور لتشكل صورة نهائية مركبة .

الصيغة العامة للأمر *glClearColor*

void glClearColor (GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);

يعين لون المسح الحالي لاستخدامه في مسح الذاكرة المؤقتة للون *Clear Color* في نمط الألوان *RGBA* (أحمر *Red* ، أخضر *Green* ، أزرق *Blue* ، قناة ألفا لتحديد الشفافية *Alpha*).

تكون قيم الأحمر والأخضر والأزرق وقناة ألفا (تحدد الشفافية) ضمن المجال *[0,1]* فمثلاً القيمة (*0,0,0,0*) تعني اللون الأسود ، والقيمة (*1,1,1,0*) تعني اللون الأبيض .

الصيغة العامة للأمر glClear

```
void glClear (GLbitfield mask);
```

تمسح الذواكر المؤقتة *buffers* ضمن الوسيط الخاص بالأمر إلى القيم الحالية المجهزة .

الوسيط *mask* عبارة عن ذاكرة مؤقتة واحدة أو أكثر من الذواكر المؤقتة *buffers* المذكورة في الجدول السابق والتي يربط بينها *OR*.يمكن فصل تعليمة *glClear* إلى عدة تعليمات بدلاً من استخدام *OR*.

تحديد الألوان

نمط الألوان في الشاشة *RGBA* ، والقيم اللونية ضمن المجال *[1 , 0]*. تجهز دائماً الألوان ثم ترسم حسب آخر لون مجهز.

يجهز الأمر *glColor3f()* لون رسم العناصر. يبين الشكل التالي بعض القيم اللونية الأساسية *RGB* و *CMYK* (سماوي *Cyan* ، بنفسجي *Magenta* ، أصفر *Yellow* ، أسود *Black*).

الأمر	اللون
glColor3f(0.0,0.0,0.0)	أسود
glColor3f(1.0,0.0,0.0)	أحمر
glColor3f(0.0,1.0,0.0)	أخضر
glColor3f(0.0,0.0,1.0)	أزرق
glColor3f(1.0,1.0,0.0)	أصفر
glColor3f(1.0,0.0,1.0)	بنفسجي
glColor3f(0.0,1.0,1.0)	سماوي
glColor3f(1.0,1.0,1.0)	أبيض

النقاط الرؤوس والخطوط والمضلعات

١ـ النقاط الرؤوس vertices

تُمثل النقطة (الرأس) بمجموعة أرقام فاصلة عائمة وتسمى *vertex*. تنفذ جميع الحسابات الداخلية كما لو كانت الرؤوس ثلاثية الأبعاد. فإذا حدد المستخدم بعدين للرأس فقط *(x,y)* فعند ذلك تسند *OpenGL* القيمة *0* للبعد الثالث *z*. تضيف *OpenGL* قيمة إحداثية رابعة *w* وذلك عند عمل *OpenGL* بالإحداثيات المتجانسة حيث يتم بواسطة تقسيم *x,y,z* على *w* الحصول على إحداثيات البعد الاقليدي (اقليدس) للنقطة *(x/w , y/w, z/w)*. *w* نادراً ما تستخدم في *OpenGL*، وبشكل افتراضي يسند *OpenGL* القيمة *1* للاحداثي *w*.

تحديد الرؤوس

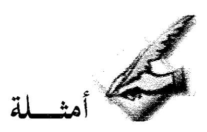
توصف في *OpenGL* جميع العناصر الهندسية كمجموعة مرتبة من الرؤوس. الصيغة العامة لأمر رسم رأس:

```
Void glVertex {234} {sifd} [v] (TYPE coords);
```

تحدد *{234}* عدد الوسائط حيث يمكن تزويد وسيطين *(x,y)* أو ثلاثة وسائط *(x,y,z)* أو أربعة وسائط *(x,y,z,w)*. القيمة الافتراضية لـــ*Z,W* هي *z=0* و *w=1* عند عدم تحديدها. استدعاء هذا الأمر يجب أن يكون بين *glBegin()* و *glEnd()*.

تمثل *{sifd}* نوع الوسائط . يمكن أن يكون النوع *short* أو *integer* أو *float* أو *double* .

يمكن أن تزود الوسائط على شكل مصفوفة من القيم (شعاع) ، ويوضع عند ذلك الحرف *v* .


أمثـــلة

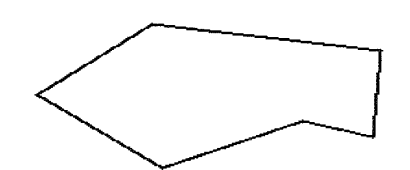
```
glVertex2s(2, 3);
glVertex3d(0.0, 0.0, 3.1415926535898);
glVertex4f(2.3, 1.0, -2.2, 2.0);
```

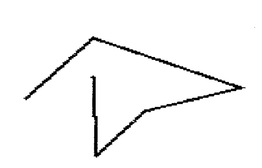
تعريف مصفوفة اسمها *dvect* بلغة ++VC وعدد عناصرها ٣

```
GLdouble dvect[3] = {5.0, 9.0, 1992.0};
glVertex3dv(dvect);
```

٢ ـ الخطوط ***Lines***

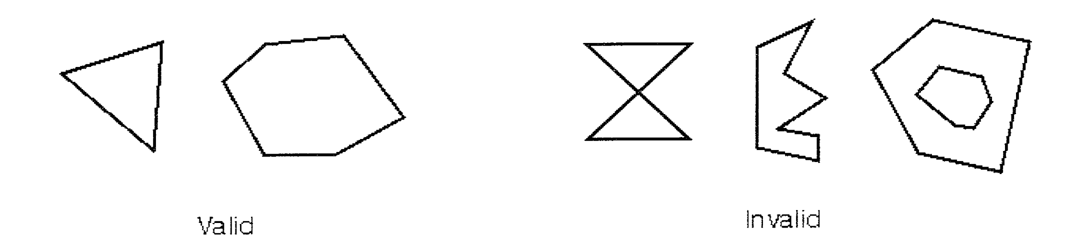
الخط في *OpenGL* عبارة عن قطعة مستقيمة، وليس كتعريف الرياضيون له أنه عبارة عن خط يمتد إلى اللانهاية من كلا الطرفين . يمكن رسم عدة خطوط متصلة وكذلك يمكن بواسطة الخطوط المتصلة رسم شكل مغلق ، يتم رسم الخطوط بتحديد رؤوس *vertices* النهايات لتلك الخطوط.





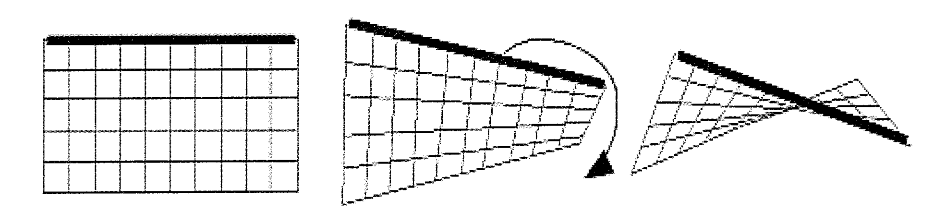
٣ ـ المضلعات ***Polygons***

المضلع عبارة عن مساحة متضمنة بواسطة حلقة مغلقة وحيدة من مقاطع الخطوط. كل حلقة مغلقة تمثل ضلعاً. يمكن أن تكون المضلعات ممتلئة (ملئ النقاط الضوئية داخلياً) كما يمكن أن ترسم بخطوط خارجية. هناك بعض القيود التي تضعها *OpenGL* لبناء مضلع قياسي (بسيط) كما هو مبين في الشكل التالي:



المضلعان في يسار الشكل صحيحان *valid* . أما المضلعات الثلاثة الأخرى فهي غير صحيحة *invalid* لأنه لا يجوز أن يكون هناك تقاطع بين الأضلاع ولا يجوز الدخول بالأضلاع نحو الداخل ولا يجوز رسم المضلعات المتداخلة .

يؤثر إسناد قيم مختلفة لرؤوس المضلع في المستوي على شكل المضلع بحيث يصبح مضلع معقد أحيانا.



٤ـ المستطيل *Rectangle*

تؤمن *OpenGL* أمر رسم مستطيل أساسي باستخدام الأمر *glRect*()*. يمكن أيضاً رسم المستطيل كمضلع (باستخدام النقاط).

الصيغة العامة لأمر رسم المستطيل

```
Void glRect {sifd} (TYPE x1, TYPE y1, TYPE x2, TYPE y2);
```

أو

```
Void glRect {sifd} v (TYPE * v1, TYPE * v2);
```



مثـــــال

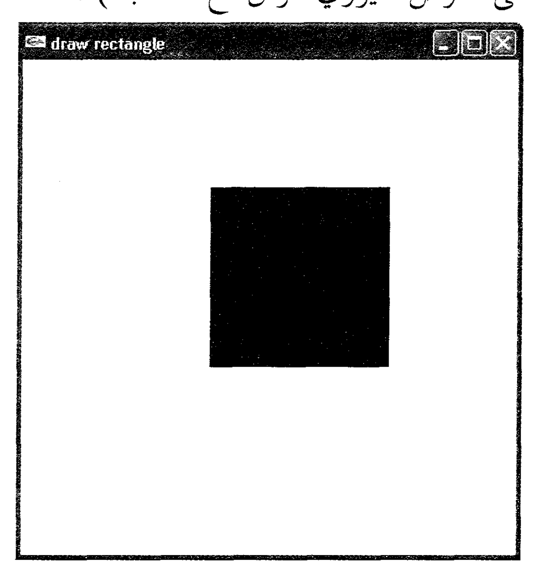
رسم مستطيل مباشرة

```
glRecti (-10,-10,20,20);
```

رسم المستطيل السابق باستخدام المصفوفات (الأشعة)

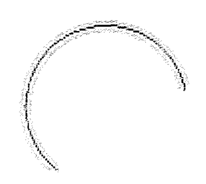
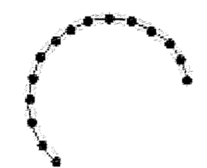
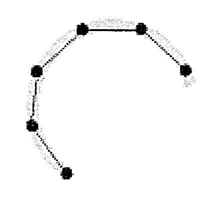
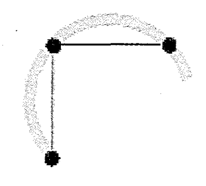
```
int r1 [] = {-10,-10};
int r2 [] = {20,20};
glRectiv (r1,r2);
```

الشكل الناتج كما يلي (مجلد هذا المثال اسمه *example21* يتوضع ضمن مجلد *examples* الموجود على القرص الليزري المرفق مع الكتاب) :



٥ـ المنحنيات *Curves*

ينتج الخط المنحني بتجميع مقاطع خطوط صغيرة .كلما ازدادت عدد المقاطع الخطية كلما ازدادت دقة الانحناء. يبين الشكل التالي ذلك :

أساسيات الرسم الهندسي في *OpenGL*

باستخدام أمر رسم الرؤوس تستطيع رسم مجموعة من النقاط أو خط أو مضلع وذلك بوضع كل مجموعة من النقاط بترتيب معين ضمن *glBegin( )* و *glEnd ( )* . يبين الشكل التالي مجموعة أوامر *glVertex* موضوعة ضمن *glBegin( )* و *glEnd ( )* . فإذا وضعنا الوسيط *GL_POLYGON* ضمن *glBegin* لتصبح *glBegin(GL_POLYGON)* يتم عند ذلك رسم مضلع مليء كما هو مبين في الشكل التالي. أما إذا وضعنا الوسيط *GL_POINTS* ضمن *glBegin* لتصبح *glBegin(GL_POINTS)* يتم عند ذلك رسم مجموعة نقاط عددها يساوي عدد أوامر *glVertex* كما هو مبين في الشكل التالي (تستطيع تجريب النص البرمجي المبين في هذا الشكل بفتح المجلد *example22* الذي يحوي هذا المثال):

```
glBegin(GL_POLYGON);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.0, 3.0);
    glVertex2f(3.0, 3.0);
    glVertex2f(4.0, 1.5);
    glVertex2f(3.0, 0.0);
glEnd();
```

GL_POLYGON GL_POINTS

الشكل العام للأمرين *glBegin( )* و *glEnd( )*

الشكل العام للأمر *glBegin( )*

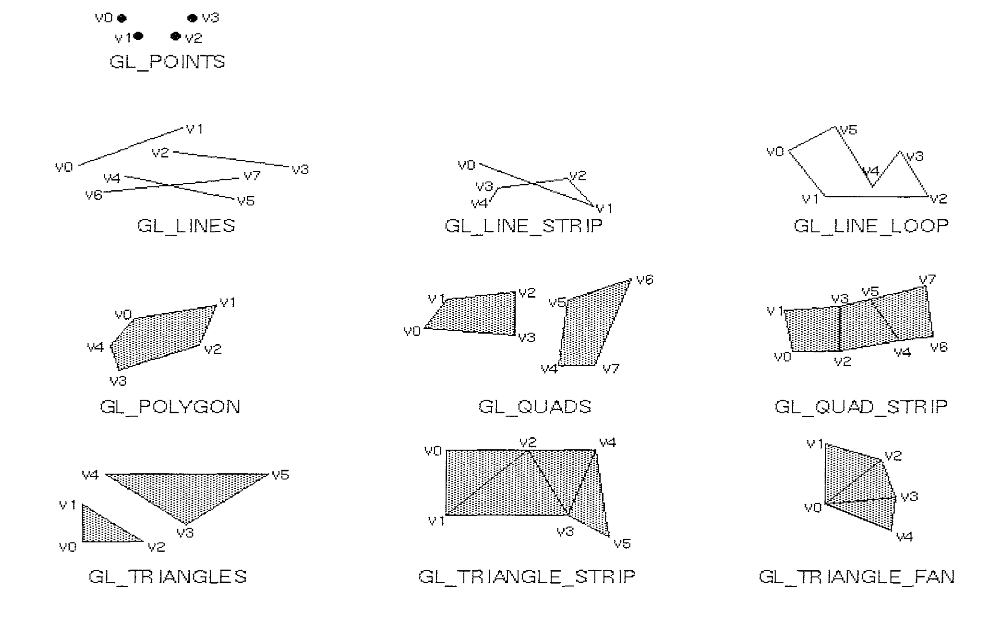
Void glBegin(GLenum mode);

يبين الجدول التالي وسائط الأمر *glBegin( )* المحتملة ونوع العنصر الهندسي المقابل .

الوسيط	المعنى
GL_POINTS	نقاط منفصلة.
GL_LINES	أزواج من الرؤوس تفسر كخطوط منفصلة.
GL_POLYGON	مضلع بسيط محدب .
GL_TRIANGLES	رؤوس ثلاثية تفسر كمثلث.

GL_QUADS	أربعة رؤوس تفسر كمضلع ذو أربع حواف .
GL_LINE_STRIP	سلسلة من الخطوط المتصلة .
GL_LINE_LOOP	نفس الوسيط السابق ولكن يضاف هنا خط بين أول وآخر رأس .
GL_TRIANGLE_STRIP	مجموعة مثلثات مستقلة موصولة إلى بعضها .
GL_TRIANGLE_FAN	مجموعة مثلثات مستقلة موصولة إلى بعضها على شكل مروحة .
GL_QUAD_STRIP	مجموعة رباعيات أضلاع مستقلة موصولة إلى بعضها .

يبين الشكل التالي أمثلة تتعلق بالعناصر الهندسية المذكورة في الجدول السابق :



لنشرح الأمثلة المبينة في الشكل السابق على افتراض أنه لدينا *n* نقطة وأن كل نقطة *(v0,v1,v2...)* تنشئ باستخدام الأمر *glVertex* :

المثال	الشرح
GL_POINTS	رسم نقطة مستقلة لكل أمر *glVertex* .
GL_LINES	رسم سلسلة مستقلة من الخطوط *v0,v1* ثم *v2,v3*

	وهكذا . إذا كانت *n* (عدد النقاط) فردي ، يتم عند ذلك تجاهل قيمة آخر نقطة .
GL_POLYGON	يتم رسم مضلع عدد رؤوسه من *v0* حتى *vn-1* . يجب أن تكون *n>=3* و في حالة عدم تحقق الشرط ، فإن هذا الأمر لا يرسم أي شيء .
GL_TRIANGLES	رسم مثلث رؤوسه *v0,v1,v2* ثم مثلث رؤوسه *v3,v4,v5*. يجب أن تكون *n* من مضاعفات العدد *3* وفي حالة عدم ذلك ، يتم تجاهل آخر نقطة أو نقطتين .
GL_QUADS	رسم مضلعات رباعية الحواف . يجب أن تكون *n* من مضاعفات العدد *4* وفي حالة عدم ذلك ، يتم تجاهل آخر نقطة أو نقطتين أو ثلاث نقاط .
GL_LINE_STRIP	رسم خطوط متصلة من *v0* إلى *v1* ثم من *v1* إلى *v2* وهكذا . يجب أن تكون *n>=1* وسنحصل في النهاية على *n-1* خط دون وصل آخر نقطة*vn-1* مع أول نقطة *v0*.
GL_LINE_LOOP	نفس الخيار السابق ، ولكن هنا يتم وصل آخر نقطة*vn-1* مع أول نقطة *v0*.
GL_TRIANGLE_STRIP	رسم سلسلة مثلثات متصلة مع بعضها البعض من المحيط باتجاه الداخل وبشكل مرتب. يتم أولاً رسم المثلث *v0,v1,v2* ثم المثلث *v2,v1,v3* ثم المثلث *v4,v3,v5* . يجب أن تكون *n>=3* .
GL_TRIANGLE_FAN	رسم سلسلة مثلثات متصلة مع بعضها البعض من المركز باتجاه المحيط وبشكل مرتب. يتم أولاً رسم المثلث *v0,v1,v2* ثم المثلث *v0,v2,v3* ثم المثلث *v0,v3,v4* . يجب أن تكون *n>=3* .
GL_QUAD_STRIP	رسم سلسلة مضلعات رباعية متصلة مع بعضها البعض .

	يتم أولاً رسم الرباعي v0,v1,v2,v3 ثم الرباعي v2,v3,v4,v5 ثم الرباعي v4,v5,v6,v7 . يجب أن تكون n>=4. إذا كانت n فردي يتجاهل آخر نقطة .

الصيغة العامة للأمر glEnd

Void glEnd(void);

يعبر هذا الأمر عن نهاية لائحة النقاط .

قيود استخدام تعليمتي glBegin() و glEnd()

تمثل المعلومات الهامة للرؤوس بالإحداثيات *glVertex()* و اللون *Color*و الشعاع الاعتيادي *normal vector* وإحداثيات التراكيب (الإكساء) *Texture coordinates* ... الخ . يحوي الجدول التالي الأوامر الممكن وقوعها يبين تعليمتي *glBegin( )* و *glEnd()*

الأمر	الغرض من الأمر
glVertex()*	يعين إحداثيات الرؤوس .
glColor()*	يعين اللون الحالي .
glIndex()*	يعين فهرس اللون الحالي.
glNormal()*	يعين إحداثيات الناظم (الشعاع الاعتيادي) .
glEvalCoord()*	يولد إحداثيات .
glCallList(), glCallLists()	ينفذ لائحة إظهار .
glTexCoord()*	يعين إحداثيات التراكيب .
glEdgeFlag()*	يضبط رسم الحواف .
glMaterial()*	يعين خصائص مواد الإكساء .

استدعاء أي أوامر أخرى بين الأمرين السابقين غير صحيح وقد يسبب حدوث أخطاء.

إظهار النقاط والخطوط والمضلعات

ترسم النقاط افتراضياً كنقطة ضوئية وحيدة على الشاشة والخط يرسم بعرض نقطة ضوئية واحدة والمضلع يرسم بشكل ممتلئ ومصمت . سنقوم بدراسة كيفية تغيير هذه الافتراضيات .

١. تفاصيل النقطة

للتحكم بحجم النقطة المصيرة ، نستخدم الأمر *glPointSize()* ونزود الحجم المرغوب بالنقطة الضوئية كوسيط للأمر السابق .

الشكل العام للأمر

```
Void glPointSize(GLfloat size);
```

علماً أن حجم النقاط يجب أن يكون أكبر من الصفر والقيمة الافتراضية هي الواحد .

٢. تفاصيل الخطوط

نستطيع مع *OpenGL* تعيين خطوط بعرض مختلف وبأنواع مختلفة (منقطة ، خطوط مقطعة ، مستمرة)

الشكل العام لأمر التحكم بعرض الخط

```
Void glLineWidth(GLfloat Width);
```

يحدد عرض الخطوط بالنقطة الضوئية ، يجب أن يكون عرض الخط أكبر من الصفر والقيمة الافتراضية هي الواحد .

الشكل العام لأمر التحكم بنوع الخط

```
Void glLine Stipple(GLint factor , GLushort pattern);
```

الوسيط *pattern* عبارة عن سلسلة بطول *16 bit* من الأصفار والواحدات . تُكرر حسب الضرورة لتحديد نوع الخط المعطى . القيمة *1* تشير لحدوث رسم و *0* لا ترسم شيء. الوسيط *Factor* يستخدم كعامل ضرب للسلسلة . فإذا تعاقبت ثلاثة واحدات

فسوف تمتد إلى ستة واحدات إذا كان *factor=2* . (قيمة *factor* بين ١ و ٢٥٥).
يُفعل الأمر السابق باستخدام *glEnable()* مع الوسيط *GL_LINE_STIPPLE* ويلغى تفعيل الأمر السابق باستخدام *glDisable( )* مع نفس الوسيط .



مثـــــال

```
glLineStipple(1,0x3f07);
glEnable(GL_LINE_STIPPLE);
```

الأمر الأول يحدد نوع الخط والأمر الثاني يفعل نوع الخط

يترجم المثال السابق كما يلي:

0x3f07 <== *0011111100000111* : ترسم الخط بثلاث نقاط ضوئية *on* ثم خمس نقاط ضوئية ***off*** ثم ست نقاط ضوئية *on* ونقطتين ضوئيتين *off* (يتم الرسم من اليمين إلى اليسار في الترميز الثنائي) .

الخانات الأقل أهمية تستخدم أولاً . أي أن *7(0111)* سترسم أولاً ثم *0(0000)* ثم *f(1111)* ثم *3(0011)* . إذا كان *factor=2* ستصبح لدينا ست نقاط ضوئية *on* وعشرة نقاط ضوئية *off* وهكذا .

يبين الشكل التالي أمثلة لأنواع خطوط وعوامل ضرب مختلفة:

PATTERN	FACTOR	
0x00FF	1	—————— —————— ——————
0x00FF	2	———————————— ————————————
0x0C0F	1	——— — ——— — ——— —
0x0C0F	3	————————— —————
0xAAAA	1	- -
0xAAAA	2	— — — — — — — — — — — —
0xAAAA	3	—— —— —— —— —— —— —— ——
0xAAAA	4	——— ——— ——— ——— ——— ———

ملاحظة	عندما يوضع الرمز 0x قبل أي رقم في لغة VC++ فهذا يعني أن هذا الرقم بالصيغة الست عشرية Hexadecimal .

٣. تفاصيل المضلعات

ترسم المضلعات ممتلئة ومغلقة ، كما يمكن أن ترسم كخطوط خارجية غير ممتلئة أو كنقاط على الرؤوس . للمضلع وجهان أمامي وخلفي وممكن تصيير كل وجه بشكل مختلف.

الشكل العام للأمر الذي يحدد نمط المضلع

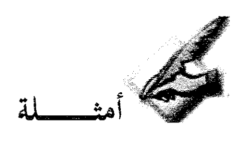
```
Void glPolygonMode(GLenum face,GLenum mode);
```

تأخذ *face* (وجه المضلع) أحد القيم التالية :

GL_BACK , GL_FRONT_AND_BACK , GL_FRONT

أما بالنسبة للنموذج *mode* (نمط الملء) فيكون :

GL_FILL ،GL_LINE ،GL_POINT


أمثـــــلة

وجه أمامي مليء (مصمت)	**glPolygonMode (GL_FRONT, GL_FILL) ;**
وجه خلفي بحدود فقط	**glPolygonMode (GL_BACK, GL_LINE) ;**

٤. نماذج ملء المضلع

افتراضياً ترسم مضلعات مليئة بنماذج مصمتة. يمكن أيضاً استخدام نماذج ملئ عبارة عن إطارات *32 bit * 32 bit* تحاذى إلى بعضها البعض.

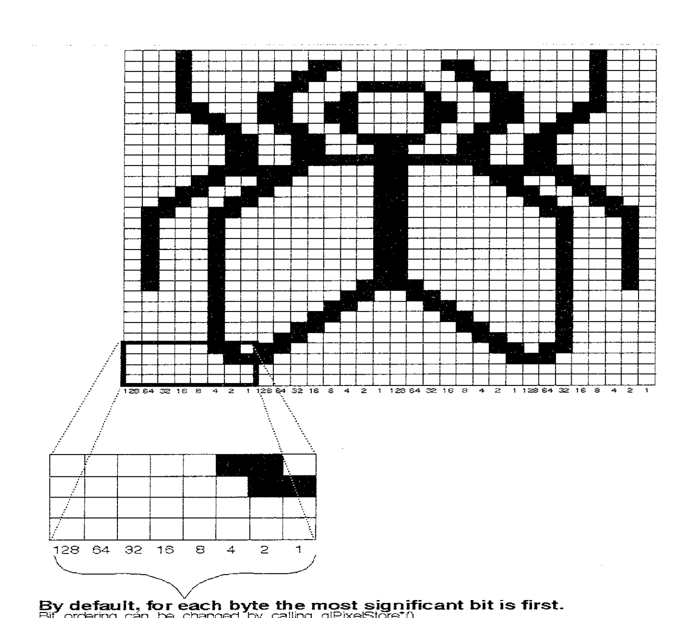
يستخدم الأمر التالي لتحديد نموذج الملء:

```
void glPolygonStipple(const GLubyte * mask);
```

الوسيط *mask* عبارة عن مؤشر إلى صورة بأبعاد *32 x 32 bit* تفسر كقناع من الأصفار والواحدات. تظهر القيمة *1* نقطة ضوئية موافقة على الشاشة، أما القيمة *0* فلا

ترسم أي شيء على الشاشة. يفعل ويعطل الأمر السابق باستخدام *glEnable()* و*glDisable()* مع الوسيط *GL_POLYGON_STIPPLE*.

يبين الشكل التالي كيفية بناء نموذج ملء (نموذج فراشة) . عدد المربعات *32*32* . يمثل المربع الأسود بالقيمة *1* والمربع الأبيض بالقيمة *0* . بإمكانك بناء نماذج مختلفة باستخدام شبكة المربعات هذه .



يتم بعد رسم النموذج السابق والتعويض بالقيم *1* و*0* ، بناء مصفوفة تعبر عن هذا النموذج حيث يتم ملء عناصر المصفوفة اعتباراً من الزاوية اليسارية السفلية للنموذج (يؤخذ سطراً ًكاملاً من المربعات وبعد الانتهاء من السطر تتم العودة إلى بداية (الزاوية اليسارية) السطر التالي وأخذ القيم وهكذا تتكرر العملية حتى انتهاء الأسطر). المصفوفة التي تعبر عن نموذج الفراشة السابق هي:

```
GLubyte fly[] = {
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x03, 0x80, 0x01, 0xC0, 0x06, 0xC0, 0x03, 0x60,
```

```
0x04, 0x60, 0x06, 0x20, 0x04, 0x30, 0x0C, 0x20,
0x04, 0x18, 0x18, 0x20, 0x04, 0x0C, 0x30, 0x20,
0x04, 0x06, 0x60, 0x20, 0x44, 0x03, 0xC0, 0x22,
0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
0x66, 0x01, 0x80, 0x66, 0x33, 0x01, 0x80, 0xCC,
0x19, 0x81, 0x81, 0x98, 0x0C, 0xC1, 0x83, 0x30,
0x07, 0xe1, 0x87, 0xe0, 0x03, 0x3f, 0xfc, 0xc0,
0x03, 0x31, 0x8c, 0xc0, 0x03, 0x33, 0xcc, 0xc0,
0x06, 0x64, 0x26, 0x60, 0x0c, 0xcc, 0x33, 0x30,
0x18, 0xcc, 0x33, 0x18, 0x10, 0xc4, 0x23, 0x08,
0x10, 0x63, 0xC6, 0x08, 0x10, 0x30, 0x0c, 0x08,
0x10, 0x18, 0x18, 0x08, 0x10, 0x00, 0x00, 0x08};
```

لاحظ أن كل سطر في المصفوفة يقابل سطرين من مربعات النموذج.

النواظم (الأشعة الاعتيادية) *Normal Vectors*

الناظم شعاع عمودي على السطح. تستطيع في *OpenGL* تحديد الناظم لكل رأس.

يستخدم الأمر *glNormal*()* لتحديد الناظم الحالي، غالباً ما يكون لكل رأس ناظم مختلف.

الشكل العام لأمر الناظم باستخدام الطريقة العادية

```
void glNormal 3 {bsidf} (TYPE nx, TYPE ny, TYPE nz);
```

الشكل العام لأمر الناظم باستخدام طريقة الأشعة

```
void glNormal 3 {bsidf}v (Const TYPE *v);
```



أمثلة

```
glBegin (GL_POLYGON);
        glNormal3fv(n0); //تمثل n0 إحداثيات الناظم
        glVertex3fv(v0); //تمثل v0 إحداثيات النقطة
        glNormal3fv(n1);
        glVertex3fv(v1);
        glNormal3fv(n2);
        glVertex3fv(v2);
        glNormal3fv(n3);
        glVertex3fv(v3);
glEnd();
```

للناظم اتجاهان متعاكسان، أحدهما للخارج والآخر للداخل ولكي ترى السطح يجب أن يكون اتجاه الناظم من السطح باتجاه الناظر ، تستطيع قلب الناظم من *(x,y,z)* إلى *(-x,-y,-z)*.تبنى السطوح اعتماداً على مضلعات صغيرة (مثلثات أو مربعات...) .

تقليل المضلعات المشكلة للسطح يسرع التصيير ولكن مظهر السطح يكون مشوهاً، وزيادتها بشكل كبير يؤدي إلى مظهر جيد ولكن بزمن تصيير كبير.

تطبيقات عملية

لتنفيذ التطبيقات التالية أنشئ تطبيقات فارغة بلغة *VC++* كما تعلمت ذلك في الفصل الأول ثم اكتب النص البرمجي المذكور في كل تطبيق .

تطبيق 1

رسم دائرة بواسطة OpenGL

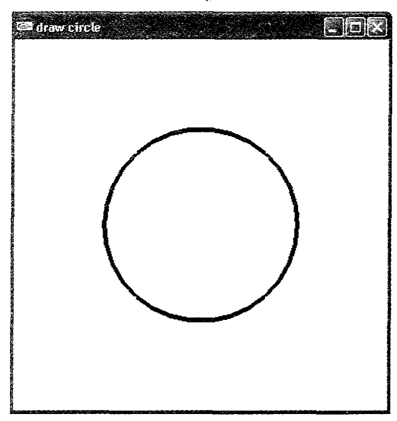
سنعمل على رسم دائرة باستخدام علاقة رياضية تعتمد على قوانين التوابع المثلثية *Sin* و*Cos* . أنشئ تطبيقاً اسمه *application21* وأنشئ ضمنه الملف *a3.cpp* ثم اكتب النص البرمجي التالي ضمن الملف *a3.cpp* :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#include <math.h> // Sin,Cos أضيفت هذه المكتبة من أجل التوابع الرياضية
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
 GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("draw circle");
 glutDisplayFunc(redraw);
```

```
  glMatrixMode(GL_PROJECTION);
  gluPerspective(45,1.0,10.0,200.0);
  glMatrixMode(GL_MODELVIEW);
  glutMainLoop();
  return 0;
}
static void redraw(void)
{
#define PI 3.14159265
#define EDGES 30 // تزداد دقة الدائرة بازدياد هذا الرقم (عدد الحواف)
#define factor 15 //عامل ضرب لتكبير الدائرة
glClearColor(1.0,1.0,1.0,0.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glColor3f(0.0,0.0,0.0);
glTranslatef(0.0,0.0,-100.0);
for (int i = 0; i < EDGES; i++) {
   glBegin(GL_LINE_LOOP);

        glVertex2f(factor*cos((2*PI*i)/EDGES),factor*sin(
        (2*PI*i)/EDGES));
        glVertex2f(factor*cos((2*PI*(i+1))/EDGES),factor
        *sin((2*PI*(i+1))/EDGES));
   glEnd(); }
glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



تطبيق 2
استخدام نماذج خطوط متنوعة

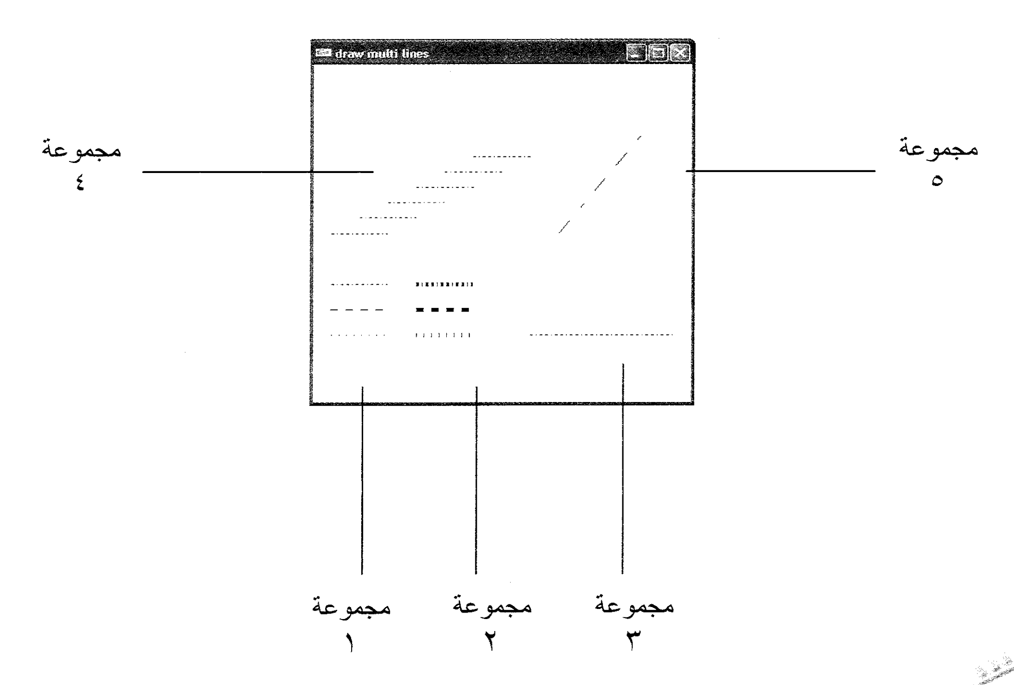
سنعمل على رسم 5 نماذج لخطوط منقطة . أنشئ تطبيقاً اسمه application22 وأنشئ ضمنه الملف a4.cpp ثم اكتب النص البرمجي التالي ضمن الملف a4.cpp :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("draw multi lines");
 glutDisplayFunc(redraw);
 glMatrixMode(GL_PROJECTION);
 gluPerspective(45,1.0,10.0,200.0);
 glMatrixMode(GL_MODELVIEW);
 glutMainLoop();
 return 0;
}
static void redraw(void)
{
 float x1,x2,y1,y2;
 int i;
 //تعريف تابع لرسم الخطوط
 #define drawOneLine(x1,y1,x2,y2) glBegin(GL_LINES);\
 glVertex2f ((x1),(y1)); glVertex2f ((x2),(y2)); glEnd();
 glClearColor(1.0,1.0,1.0,0.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
 glTranslatef(0.0f,0.0f,-100.0f);
```

```
/*  رسم جميع الخطوط باللون الأبيض  */
glColor3f (0.0, 0.0, 0.0);
/*  تتألف المجموعة١ من ٣ خطوط بأنواع مختلفة  */
glEnable (GL_LINE_STIPPLE);
glLineStipple (1, 0x0101);  /*  نوع الخط منقط  */
drawOneLine (-30.0, -20.0, -20.0, -20.0);
glLineStipple (1, 0x00FF);  /*  نوع الخط شرطة سفلية  */
drawOneLine (-30.0, -15.0, -20.0, -15.0);
glLineStipple (1, 0x1C47); /*  نوع الخط شرطة سفلية – نقطة – شرطة سفلية  */
drawOneLine (-30.0, -10.0, -20.0, -10.0);
/*  تتألف المجموعة٢ من ٣ خطوط عريضة بأنواع مختلفة  */
glLineWidth (5.0);//عرض الخط ٥
glLineStipple (1, 0x0101);
drawOneLine (-15.0, -20.0, -5.0, -20.0);
glLineStipple (1, 0x00FF);
drawOneLine (-15.0, -15.0, -5.0, -15.0);
glLineStipple (1, 0x1C47);
drawOneLine (-15.0, -10.0, -5.0, -10.0);
glLineWidth (1.0);
/*  تتألف المجموعة٣ من ٦ خطوط نوع الخط شرطة سفلية – نقطة – شرطة سفلية  */
/*  تشكل جزءً من خط طويل واحد  */
glLineStipple (1, 0x1C47);
glBegin (GL_LINE_STRIP);
      for (i = 0; i <6; i++)
        glVertex2f (5.0 + ((GLfloat) i/2 * 10), -20.0);
glEnd ();
/*  تتألف المجموعة ٤ من ٦ خطوط مستقلة  */
/*  نوع الخط شرطة سفلية – نقطة – شرطة سفلية  */
for (i = 0; i < 6; i++) {
  drawOneLine (-30.0 + ((GLfloat) i * 5.0),
             i*3.0, -25.0 + ((GLfloat)(i+1) * 5.0), i*3.0);
                  }
```

```
/*  تتألف المجموعة٥ من خط واحد نوعه  شرطة سفلية – نقطة – شرطة سفلية  */
/*  وعامل التكرار ٥  */
glLineStipple (5, 0x1C47);
drawOneLine (10.0, 0.0, 25.0, 20.0);
glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



تطبيق 3

استخدام نماذج ملئ المضلعات

سنعمل على رسم 3 مستطيلات بنماذج ملء مختلفة . أنشئ تطبيقاً اسمه *application23* وأنشئ ضمنه الملف *a5.cpp* ثم اكتب النص البرمجي التالي ضمن الملف : *a5.cpp*

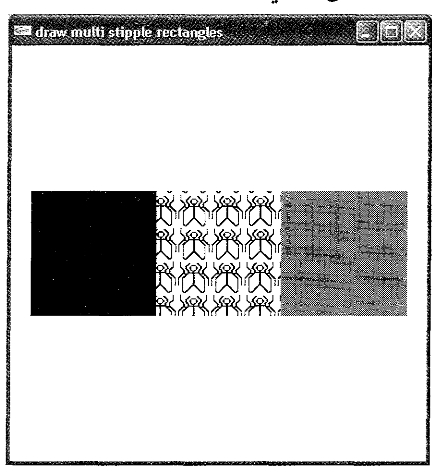
```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
```

```
#include <gl\glut.h>
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
 GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("draw multi stipple rectangles ");
 glutDisplayFunc(redraw);
 glMatrixMode(GL_PROJECTION);
 gluPerspective(45,1.0,10.0,200.0);
 glMatrixMode(GL_MODELVIEW);
 glutMainLoop();
 return 0;
}
static void redraw(void)
{
 //تعريف مصفوفة نموذج ملء الفراشة
GLubyte fly[] = {
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x03, 0x80, 0x01, 0xC0, 0x06, 0xC0, 0x03, 0x60,
      0x04, 0x60, 0x06, 0x20, 0x04, 0x30, 0x0C, 0x20,
      0x04, 0x18, 0x18, 0x20, 0x04, 0x0C, 0x30, 0x20,
      0x04, 0x06, 0x60, 0x20, 0x44, 0x03, 0xC0, 0x22,
      0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
      0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
      0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
      0x66, 0x01, 0x80, 0x66, 0x33, 0x01, 0x80, 0xCC,
      0x19, 0x81, 0x81, 0x98, 0x0C, 0xC1, 0x83, 0x30,
      0x07, 0xe1, 0x87, 0xe0, 0x03, 0x3f, 0xfc, 0xc0,
      0x03, 0x31, 0x8c, 0xc0, 0x03, 0x33, 0xcc, 0xc0,
      0x06, 0x64, 0x26, 0x60, 0x0c, 0xcc, 0x33, 0x30,
      0x18, 0xcc, 0x33, 0x18, 0x10, 0xc4, 0x23, 0x08,
      0x10, 0x63, 0xC6, 0x08, 0x10, 0x30, 0x0c, 0x08,
      0x10, 0x18, 0x18, 0x08, 0x10, 0x00, 0x00, 0x08};
```

```
//تعريف مصفوفة نموذج ملء آخر
GLubyte halftone[] = {
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55};

glClearColor(1.0,1.0,1.0,0.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glTranslatef(5.0f,15.0f,-40.0f);
glColor3f(0.0, 0.0, 0.0);
glRectf (-20.0, -20.0, -10.0, -10.0);
glEnable (GL_POLYGON_STIPPLE);
glPolygonStipple (fly);
glRectf (-10.0, -20.0, 0.0, -10.0);
glPolygonStipple (halftone);
glRectf (0.0, -20.0, 10.0, -10.0);
glDisable (GL_POLYGON_STIPPLE);
glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



تطبيق 4

تحويل المربع والمثلث من 2D (ثنائي البعد) إلى 3D (ثلاثي البعد)

سنعمل على تحويل المثلث والمربع المرسومين في التطبيق 2 من الفصل الأول إلى هرم ومكعب . أنشئ تطبيقاً اسمه *application24* وأنشئ ضمنه الملف *a6.cpp* ثم اكتب النص البرمجي التالي ضمن الملف *a6.cpp* :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowPosition(100,100);
  glutInitWindowSize(400,400);
  glutCreateWindow("draw payamid & cube");
  glutDisplayFunc(redraw);
  glMatrixMode(GL_PROJECTION);
  gluPerspective(45,1.0,10.0,200.0);
  glMatrixMode(GL_MODELVIEW);
  glutMainLoop();
  return 0;
}
static void redraw(void)
```

```
{
 glClearColor(1.0,1.0,1.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glTranslatef(-15.0f,0.0f,-100.0f);
 glRotatef(30,0.0f,1.0f,0.0f);     // Y تدوير الهرم ٣٠درجة حول المحور
 glBegin(GL_TRIANGLES);            // بدء رسم المثلث
 glColor3f(1.0f,0.0f,0.0f);        // أحمر
 glVertex3f( 0.0f, 10.0f, 0.0f); // النقطة العلوية في المثلث –الواجهة الأمامية للهرم
 glColor3f(0.0f,10.0f,0.0f);       // أخضر
 glVertex3f(-10.0f,-10.0f, 10.0f); // النقطة اليسارية في المثلث –الواجهة الأمامية للهرم
 glColor3f(0.0f,0.0f,10.0f);       // أزرق
 glVertex3f( 10.0f,-10.0f, 10.0f); // النقطة اليمينية في المثلث –الواجهة الأمامية للهرم
 glColor3f(10.0f,0.0f,0.0f);       // أحمر
 glVertex3f( 0.0f, 10.0f, 0.0f);  // النقطة العلوية في المثلث –الواجهة اليمينية للهرم
 glColor3f(0.0f,0.0f,10.0f);       // أزرق
 glVertex3f( 10.0f,-10.0f, 10.0f);// النقطة اليسارية في المثلث –الواجهة اليمينية للهرم
 glColor3f(0.0f,10.0f,0.0f);       // أخضر
 glVertex3f( 10.0f,-10.0f, -10.0f); // النقطة اليمينية في المثلث –الواجهة اليمينية للهرم
 glColor3f(10.0f,0.0f,0.0f);       // أحمر
 glVertex3f( 0.0f, 10.0f, 0.0f);  // النقطة العلوية في المثلث –الواجهة الخلفية للهرم
 glColor3f(0.0f,1.0f,0.0f);              // أخضر
 glVertex3f( 10.0f,-10.0f, -10.0f); // النقطة اليسارية في المثلث –الواجهة الخلفية للهرم
 glColor3f(0.0f,0.0f,10.0f);       // أزرق
 glVertex3f(-10.0f,-10.0f, -10.0f); // النقطة اليمينية في المثلث –الواجهة الخلفية للهرم
 glColor3f(10.0f,0.0f,0.0f);             // أحمر
 glVertex3f( 0.0f, 10.0f, 0.0f);  // النقطة العلوية في المثلث –الواجهة اليسارية للهرم
 glColor3f(0.0f,0.0f,10.0f);             // أزرق
```

```
glVertex3f(-10.0f,-10.0f,-10.0f); // النقطة اليسارية في المثلث -الواجهة اليسارية للهرم
glColor3f(0.0f,10.0f,0.0f);          // أخضر
glVertex3f(-10.0f,-10.0f, 10.0f); // النقطة اليمينية في المثلث -الواجهة اليسارية للهرم
glEnd();                              // نهاية رسم الهرم
glLoadIdentity();                          // إعادة تهيئة مصفوفة التحويلات
glTranslatef(20.0f,0.0f,-100.0f);
glRotatef(30,1.0f,1.0f,0.0f);
glBegin(GL_QUADS);         // رسم المكعب
glColor3f(0.0f,1.0f,0.0f);      // أخضر
glVertex3f( 10.0f, 10.0f,-10.0f); // النقطة العلوية اليمينية في المربع-الواجهة العلوية للمكعب
glVertex3f(-10.0f, 10.0f,-10.0f);// النقطة العلوية اليسارية في المربع-الواجهة العلوية للمكعب
glVertex3f(-10.0f, 10.0f, 10.0f);// النقطة السفلية اليسارية في المربع-الواجهة العلوية للمكعب
glVertex3f( 10.0f, 10.0f, 10.0f); // النقطة السفلية اليمينية في المربع-الواجهة العلوية للمكعب
glColor3f(1.0f,0.5f,0.0f);      // برتقالي
glVertex3f( 10.0f,-10.0f, 10.0f);// النقطة العلوية اليمينية في المربع-الواجهة السفلية للمكعب
glVertex3f(-10.0f,-10.0f, 10.0f);// النقطة العلوية اليسارية في المربع-الواجهة السفلية للمكعب
glVertex3f(-10.0f,-10.0f,-10.0f);// النقطة السفلية اليسارية في المربع-الواجهة السفلية للمكعب
glVertex3f( 10.0f,-10.0f,-10.0f);// النقطة السفلية اليمينية في المربع-الواجهة السفلية للمكعب
glColor3f(1.0f,0.0f,0.0f);      // أحمر
glVertex3f( 10.0f, 10.0f, 10.0f); // النقطة العلوية اليمينية في المربع-الواجهة الأمامية للمكعب
glVertex3f(-10.0f, 10.0f, 10.0f);//  النقطة العلوية اليسارية في المربع-الواجهة الأمامية للمكعب
glVertex3f(-10.0f,-10.0f, 10.0f);//  النقطة السفلية الياسرية في المربع-الواجهة الأمامية للمكعب
glVertex3f( 10.0f,-10.0f, 10.0f);//  النقطة السفلية اليمينية في المربع-الواجهة الأمامية للمكعب
glColor3f(1.0f,1.0f,0.0f);            // أصفر
glVertex3f( 10.0f,-10.0f,-10.0f);//  النقطة العلوية اليمينية في المربع-الواجهة الخلفية للمكعب
glVertex3f(-10.0f,-10.0f,-10.0f);//  النقطة العلوية اليسارية في المربع-الواجهة الخلفية للمكعب
glVertex3f(-10.0f, 10.0f,-10.0f);//  النقطة السفلية اليسارية في المربع-الواجهة الخلفية للمكعب
glVertex3f( 10.0f, 10.0f,-10.0f);//  النقطة السفلية اليمينية في المربع-الواجهة الخلفية للمكعب
glColor3f(0.0f,0.0f,1.0f);           // أزرق
```

```
glVertex3f(-10.0f, 10.0f, 10.0f);// النقطة العلوية اليمينية في المربع-الواجهة اليسارية للمكعب
glVertex3f(-10.0f, 10.0f,-10.0f);// النقطة العلوية اليسارية في المربع-الواجهة اليسارية للمكعب
glVertex3f(-10.0f,-10.0f,-10.0f);// النقطة السفلية اليسارية في المربع-الواجهة اليسارية للمكعب
glVertex3f(-10.0f,-10.0f, 10.0f);// النقطة السفلية اليمينية في المربع-الواجهة اليسارية للمكعب
glColor3f(1.0f,0.0f,1.0f);              // بنفسجي
glVertex3f( 10.0f, 10.0f,-10.0f);// النقطة العلوية اليمينية في المربع-الواجهة اليمينية للمكعب
glVertex3f( 10.0f, 10.0f, 10.0f); // النقطة العلوية اليسارية في المربع-الواجهة اليمينية للمكعب
glVertex3f( 10.0f,-10.0f, 10.0f);// النقطة السفلية اليسارية في المربع-الواجهة اليمينية للمكعب
glVertex3f( 10.0f,-10.0f,-10.0f);// النقطة السفلية اليمينية في المربع-الواجهة اليمينية للمكعب
glEnd();                                // نهاية رسم المكعب
glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



الفصل الثالث

الرؤية و الإظهار

ستتعلم في هذا الفصل المواضيع التالية:

- مبدأ عمل الكاميرا
- تحويلات ***modeling***
- تحويلات ***viewing***
- تحويل الإسقاط ***projection***
- تحويل ***viewport***
- سطوح قطع إضافية
- تطبيقات عملية

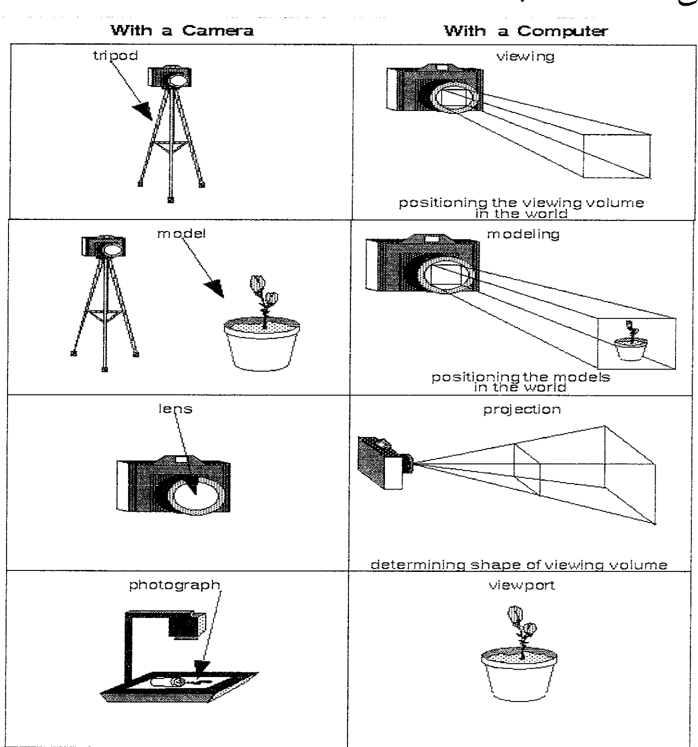
يجب تذكر أن هدف رسوميات الحاسب يتمثل في خلق صورة ثنائية البعد لأجسام ثلاثية البعد (لها بعدين فقط لأنها سترسم على الشاشة).

مبدأ عمل الكاميرا

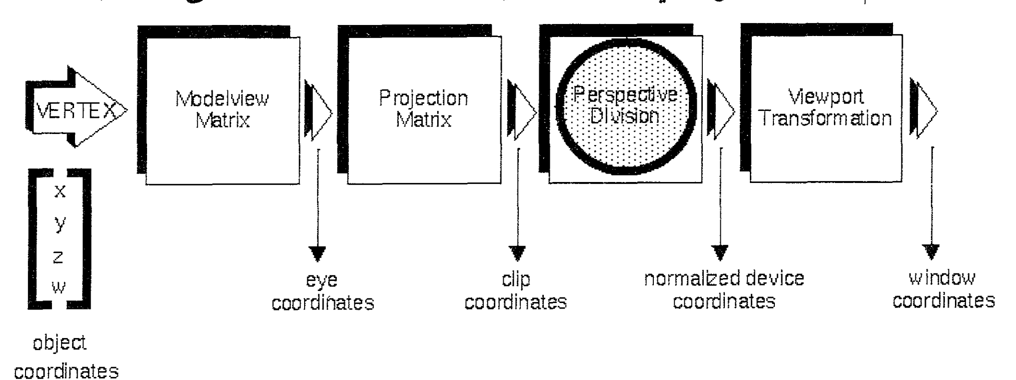
يكمن تشبيه عملية التحويل في الحاسب للحصول على المشهد المطلوب رؤيته بعملية التقاط صورة بالكاميرا. والخطوات هي:

- تثبيت حامل الكاميرا وتوجيه الكاميرا إلى المشهد (تحويلات *viewing*).
- تنظيم المشهد ووضع عناصره في المكان المناسب (تحويلات *modeling*).
- اختيار عدسة الكاميرا وتعديل التقريب والتبعيد *zoom* (تحويلات *projection*).
- تحديد حجم الصورة النهائية (تحويلات *viewport*).

بعد تنفيذ هذه الخطوات يمكن التقاط الصورة بواسطة الكاميرا وبشكل مشابه يمكن رسم المشهد على شاشة الحاسب.



هناك ترتيب لهذه العمليات، فتحويلات *viewing* يجب أن تسبق تحويلات *modeling* في شيفرة البرنامج، أما تحويلات *projection* و *viewport* فيمكن تحديدها في أي مكان قبل حدوث الرسم. يبين الشكل التالي الترتيب الذي تنفذ فيه هذه العمليات على الحاسب.





لتحقيق تحويلات *projection* و*modeling* و*viewing* ، سنبني مصفوفة *M* أبعادها *4x4* يتم ضربها بإحداثيات كل رأس *V* في المشهد لتحقيق التحويل المطلوب. موقع الرأس الجديد *V`* يساوي ناتج ضرب مصفوفة التحويل *M* بموقع الرأس القديم *V* .

$V` = MV$



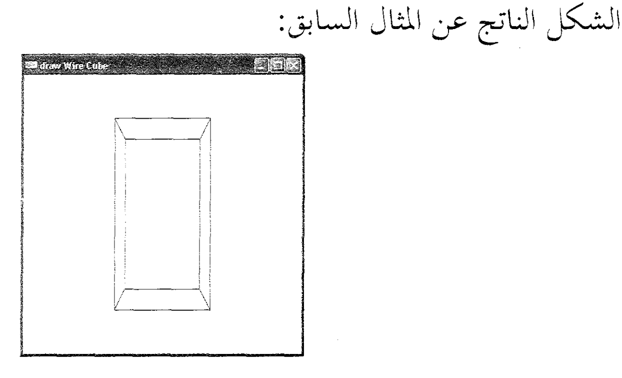
مثـــــال

يرسم هذا المثال مكعب يتم تغيير حجمه عن طريق تحويل *modeling*، ويستخدم تحويل *viewing* لنقله مسافة بسيطة عكس المحور *Z*. كذلك يتم تحديد تحويل إسقاط وتحويل *viewport*(اسم مجلد المثال *example31* ضمن المجلد *examples*) .

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#include <gl\glaux.h>
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
int h=300,w=300;
```

```
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
GLUT_DEPTH);
glutInitWindowPosition(100,100);
glutInitWindowSize(400,400);
glutCreateWindow("draw Wire Cube");
glutDisplayFunc(redraw);
glMatrixMode(GL_PROJECTION);/*  projection تعريف تحويل الاسقاط */
gluPerspective(45,1.0,10.0,200.0);
glMatrixMode(GL_MODELVIEW);
glViewport (0, 0, w, h);      /*  viewport تعريف تحويل */
glutMainLoop();
return 0;
}
static void redraw(void)
{
glClearColor(1.0,1.0,1.0,0.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glColor3f (0.0, 0.0, 0.0);
glLoadIdentity ();
glTranslatef (0.0, 0.0, -100.0); /* viewing تحويل */
glScalef (1.0, 2.0, 1.0);      /* modeling تحويل */
auxWireCube(25.0);   /*  رسم مكعب سلكي */
glutSwapBuffers();
}
```

الشكل الناتج عن المثال السابق:

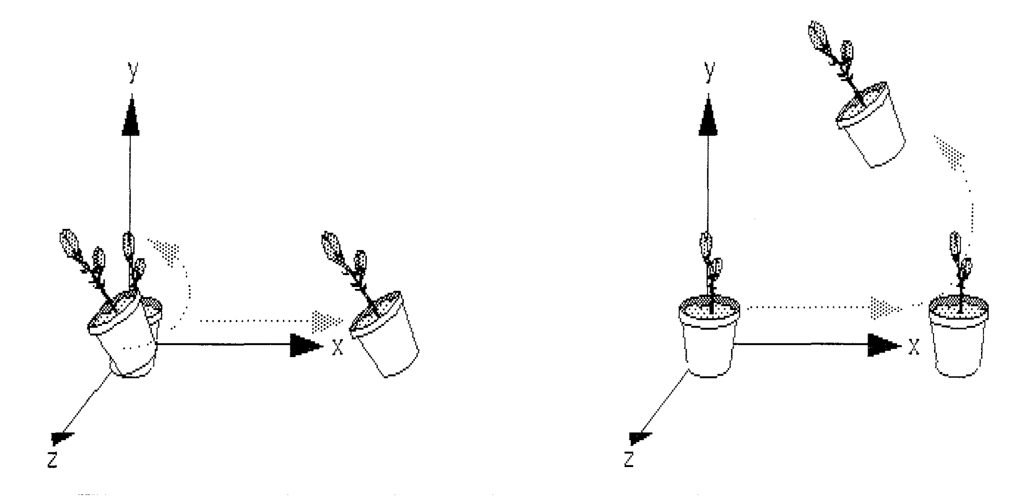


تحويلات viewing و modeling

قبل البدء بالتفاصيل تذكر أنه يجب استدعاء الأمر *glMatrixMode()* مع المعامل *GL_MODELVIEW* قبل تنفيذ تحويلات *viewing* أو *modeling* .

التفكير بالتحويلات

ترتيب التحويلات ضروري جداً. وقد يؤدي تغيير ترتيب التحويلات إلى تغيير موقع العنصر كما هو مبين في الشكل التالي.



في الشكل اليساري تم عمل دوران *45* درجة عكس عقارب الساعة ثم إزاحة وفق المحور *x* . أما في الشكل اليميني فقد أنجزت الإزاحة وفق المحور *x* أولاً ثم عمل دوران *45* درجة عكس عقارب الساعة . لاحظ اختلاف الشكلين .

لنكتب النص البرمجي للشكل السابق على افتراض أن الدوران يتم أولاً ثم التحريك:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(T);          /* التحريك */
glMultMatrixf(R);          /* الدوران */
draw_the_object();
```

تمثل تحويلات viewing و modeling بمصفوفة 4x4. آخر تحويل يستدعى في برنامجك هو فعلياً أول تحويل يطبق على الرؤوس.


مثـــــال

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(N);            /* تطبيق التحويل N */
glMultMatrixf(M);            /*تطبيق التحويل M */
glMultMatrixf(L);            /*تطبيق التحويل L */
glBegin(GL_POINTS);
  glVertex3f(v);             /*رسم الرأس v */
glEnd();
```

المصفوفة الناتجة (إحداثيات الرأس الجديد)هي: N(M(LV))

تصدر أوامر التحويل viewing أولاً ثم modeling .

تحويلات modeling

هناك ثلاثة تحويلات لـــ modeling وهي glTranslate*() و glRotate*() و glScale*().

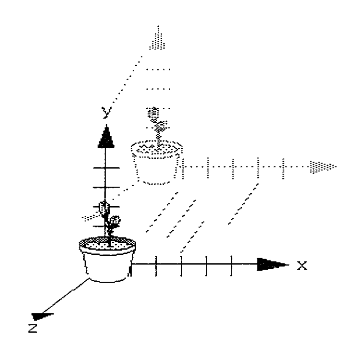
هذه التحويلات الثلاثة مكافئة لتشكيل مصفوفة النقل و الدوران و تغيير الحجم ثم استدعاء glMultMatrix*() مع المصفوفة الملائمة كوسيط.

تحويل النقل Translate

يعمل هذا التحويل على نقل أي عنصر هندسي وفق محور أو مستوي إحداثيات . الشكل العام لهذا الأمر:

```
void glTranslate{fd}(TYPEx, TYPE y, TYPEz);
```

تضرب المصفوفة الحالية بمصفوفة تنقل الجسم بمقدار قيم x,y,z (أو تحرك جملة الإحداثيات بنفس المقدار) كما هو مبين في الشكل التالي:



مصفوفة النقل الناتجة T :

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

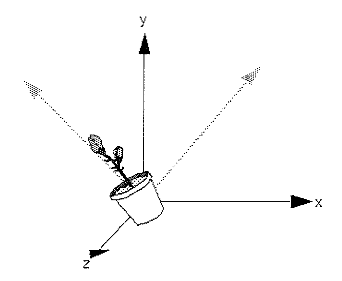
تحويل الدوران *Rotate*

يعمل هذا التحويل على تدوير أي عنصر هندسي وفق محور أو مستوي إحداثيات .

الشكل العام لهذا الأمر:

```
void glRotate{fd}(TYPE angle, TYPE x, TYPE y, TYPE z);
```

يضرب المصفوفة الحالية بمصفوفة تدور الجسم (أو نظام الإحداثيات المحلية) مع عقارب الساعة أو بعكسها حول محور معين وبزاوية تقدر بالدرجات. يبين الشكل التالي تأثير الأمر *glRotate(45.0,0.0,0.0,1.0)*



مصفوفة الدوران الناتجة R :

$$\text{glRotate*}(\alpha, 1, 0, 0): \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos a & -\sin a & 0 \\ 0 & \sin a & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate*}(\alpha, 0, 1, 0): \begin{bmatrix} \cos a & 0 & \sin a & 0 \\ 0 & 1 & 0 & 0 \\ -\sin a & 0 & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate*}(\alpha, 0, 0, 1): \begin{bmatrix} \cos a & -\sin a & 0 & 0 \\ \sin a & \cos a & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

تحويل تغيير الحجم *Scale*

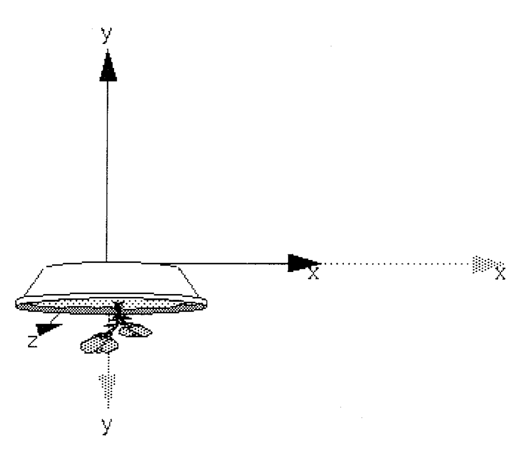
يعمل هذا التحويل على تغيير حجم أي عنصر هندسي وفق محور أو مستوي إحداثيات . الشكل العام لهذا الأمر:

```
void glScale{fd}(TYPEx, TYPE y, TYPEz);
```

يضرب المصفوفة الحالية بمصفوفة تكبر أو تمدد أو تقلص أو تعكس الجسم عبر المحاور.

يضرب كل إحداثي *x,y,z* لكل نقطة بالمعامل الموافق *x,y,z*. والجسم المرتبط يتمدد معها.

يظهر الشكل التالي تأثير التحويل *glScalef (2.0,-0.5,1.0)*



مصفوفة تغيير الحجم الناتجة *S*

$$S = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } S^{-1} = \begin{bmatrix} \frac{1}{x} & 0 & 0 & 0 \\ 0 & \frac{1}{y} & 0 & 0 \\ 0 & 0 & \frac{1}{z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

القيم أكبر من *1* للتكبير وأصغر من *1* للتصغير. القيم السالبة تعني عكس الجسم على المحور.

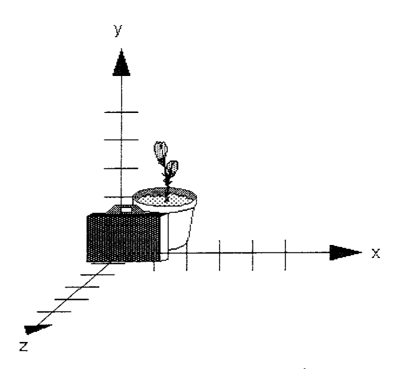
تحويلات *viewing*

تستخدم لتغيير موقع ووجهة نظر المشهد . يشبه ذلك وضع أرجل الكاميرا وتوجيهها باتجاه الجسم. تتألف عادة تحويلات *viewing* من نقل ودوران.

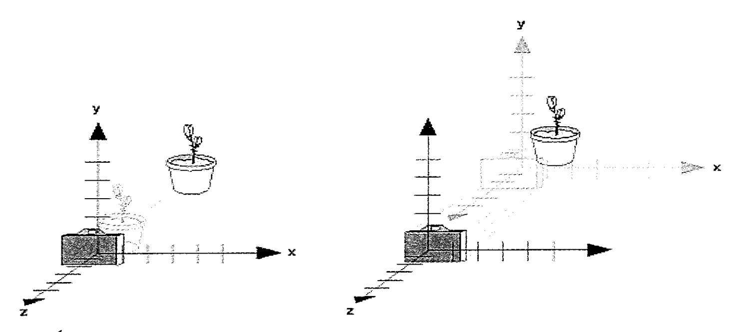
يكافئ تحويل *modeling* الذي يدور جسم عكس عقارب الساعة تحويل *viewing* الذي يدور الكاميرا مع عقارب الساعة. تذكر استدعاء أوامر تحويل *viewing* قبل تحويل *modeling*، وذلك حتى يطبق تأثير تحويلات *modeling* أولاً على البرنامج.

استخدام ()*glTranslate و ()*glRotate

توجه الكاميرا عادة نحو المحور z السالب (نرى ظهر الكاميرا) كما هو مبين في الشكل التالي:



تستطيع تحريك الكاميرا بعيداً عن الأجسام إلى الوراء وهذا له نفس تأثير تحريك الأجسام بعيداً عن الكاميرا للأمام. في القسم اليساري من الشكل التالي حركنا العنصر للداخل (المحور z السالب) باستخدام الأمر *glTranslatef(0.0,0.0,-5.0)* ، أما في القسم اليميني من نفس الشكل فقد حركنا الكاميرا للخارج (المحور z الموجب) باستخدام الأمر *glTranslatef(0.0,0.0,5.0)* :



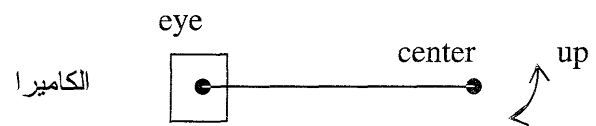
إذا أردنا رؤية العناصر من الجانب،يجب تدوير الجسم ثم تحريكه بعيداً عن الكاميرا حتى يمكن رؤية الجانب المطلوب من العنصر (تكتب الأوامر بعكس الترتيب التحريك ثم الدوران)

```
glTranslatef(0.0,0.0,-5.0)
glRotate(90.0,0.0,1.0,0.0);
```

استخدام التابع *gluLookAt()*

يستخدم هذا التابع خط الرؤية (خط أفق الكاميرا). وله الشكل العام التالي:

```
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,
GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble
upx, GLdouble upy, GLdouble upz);
```



تحدد الإحداثية *eyex, eyey, eyez* مركز الكاميرا (مركز عين الناظر) . أما الإحداثية *centerx, centery, centerz* فتحدد نقطة تقع على خط الأفق تمثل مركز رؤية الكاميرا (وجهة نظر العين) . تمثل الإحداثية *upx, upy* الاتجاه في الفضاء لأعلى وأسفل .

تحويل الإسقاط *Projection*

تستخدم تحويلات الإسقاط لتحويل الرؤوس في المشهد. قبل إصدار أي أمر تحويل اسقاط يجب تنفيذ التالي:

```
glMatrixMode( GL_PROJECTION);
glLoadIdentity();
```

لجعل تأثير التعليمات الحالية يطبق على مصفوفة الإسقاط وليس على مصفوفة *ModelView* .

يستخدم تحويل الإسقاط لتعريف فضاء الرؤية ويعرف بطريقتين:

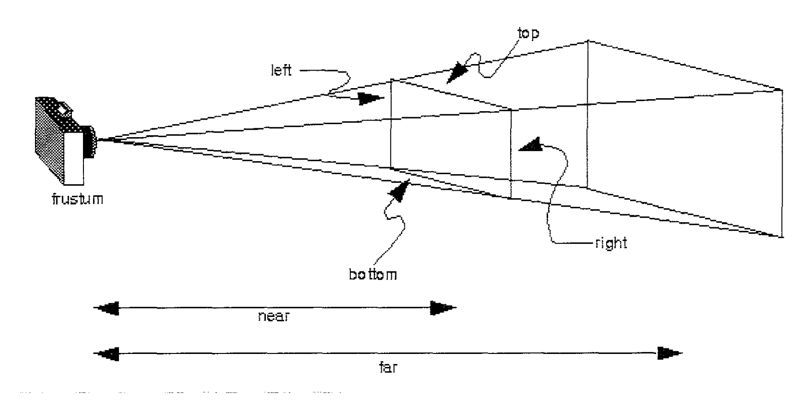
الإسقاط المنظوري *perspective projection* والإسقاط المتعامد *orthographic projection* .

١. الإسقاط المنظوري *Perspective Projection*

كلما كان الجسم أبعد عن الكاميرا، كلما ظهر أصغر. يحدث ذلك لأن حجم الإظهار للإسقاط المنظوري عبارة عن جزء من هرم (جذع هرم) بدون رأس. الأجسام الأقرب من

عين الناظر (الكاميرا) تبدو أكبر لأنها تحتل مساحة أكبر من حجم الإظهار. تستخدم هذه الطريقة من الإسقاط في الحركة وفي المحاكاة البصرية والتطبيقات الواقعية.

أمر تعريف جذع الهرم *glFrustum()* الذي يستخدم لحساب مصفوفة تنجز الإسقاط المنظوري وتضرب المصفوفة الحالية بها، وتقص الأجسام الواقعة خارج جذع الهرم.

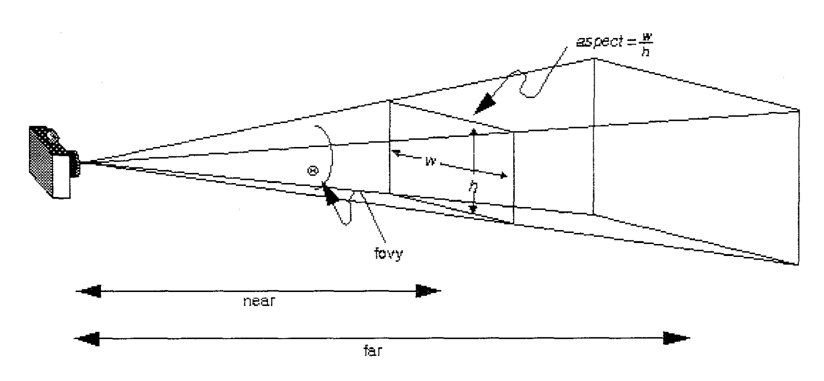


الصيغة العامة للأمر *glFrustum()* :

void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);

باستخدام قيم *left,right,bottom,top,near,far* تستطيع تحديد جذع الهرم الذي ترى العناصر ضمنه .

هناك تابع آخر *gluPerspective()* الذي يحدد زاوية حقل الرؤية في المستوي *xz* ونسبة الطول للعرض *x/y* كما في الشكل :

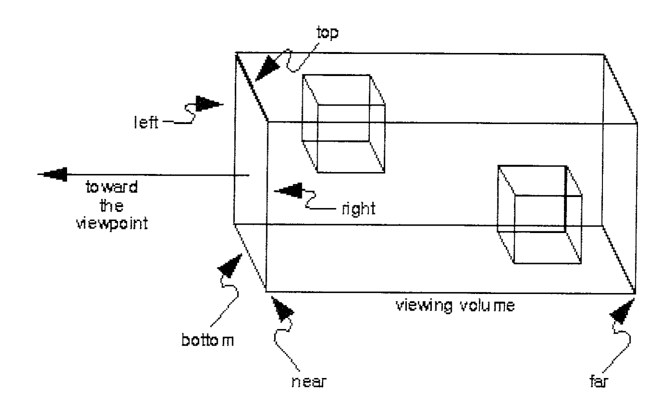


الصيغة العامة للأمر *gluPerspective()*:

```
void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar);
```

٢. الإسقاط المتعامد (على جملة إحداثيات) Orthographic Projection

فراغ الإظهار هنا عبارة عن صندوق . لا تؤثر بعد المسافة عن الكاميرا على حجم الجسم . يستخدم هذا النوع من الإسقاط لإنشاء الخرائط المعمارية والتصاميم باستخدام الحاسب .

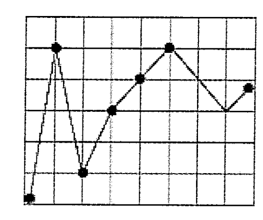


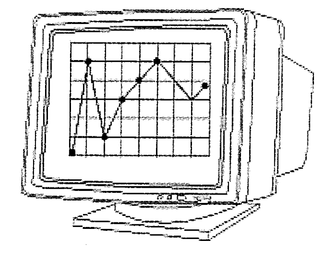
التابع المستخدم في هذا النوع من الإسقاط هو *glOrtho()* الذي له الشكل العام التالي:

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
```

تحويل viewport

viewport عبارة عن منطقة مستطيلة من الإطار يتم فيها رسم الصورة .





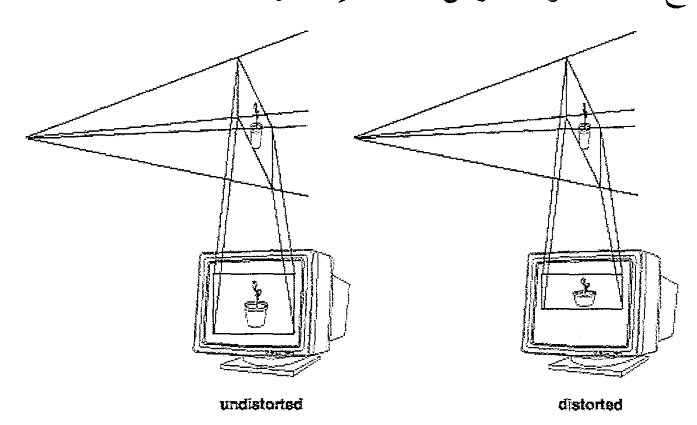
تعريف *viewport*

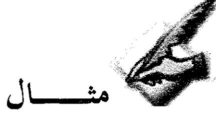
افتراضياً تشكل *viewport* كامل النقاط الضوئية لإطار الرسم. سنستخدم الأمر *glViewPort()* لاختيار منطقة رسم أصغر. وبذلك نستفيد من الإطار الواحد في الحصول على عدة مشاهد.

الشكل العام لأمر *glViewPort()*:

```
void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);
```

نسبة الطول للعرض لـ *viewport* يجب أن تكون متساوية مع نسبة الطول للعرض بالنسبة لفضاء الإظهار والإ ستشوه الصورة .يبين الشكل التالي صورة غير مشوهة (القسم اليساري) وصورة مشوهة (القسم اليميني) نتيجو اختلاف نسبة الطول للعرض لـ *viewport* مع نسبة الطول للعرض لفضاء الإظهار .





مثـــــال

الوضع الافتراضي (العرض *w* والارتفاع *h*)

```
gluPerspective(myFovy, 1.0, myNear, myFar); //w/h=1➔ w=h
glViewport(0, 0, 400, 400);// w=400 ,h=400
```

الصورة مشوهة لأنها مضغوطة وفق المحور *x*

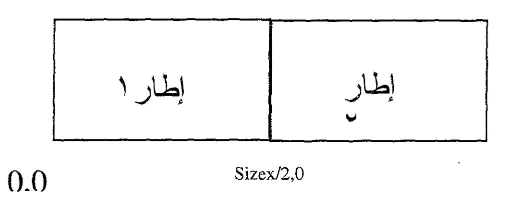
```
gluPerspective(myFovy, 2.0, myNear, myFar); // w/h=2➔ w=2h
glViewport (0, 0, 400, 400);// w=400 ,h=400
```

لتجنب التشوه

```
glViewport(0, 0, 400, 200);//w=400 , h=200, 2h=400
```

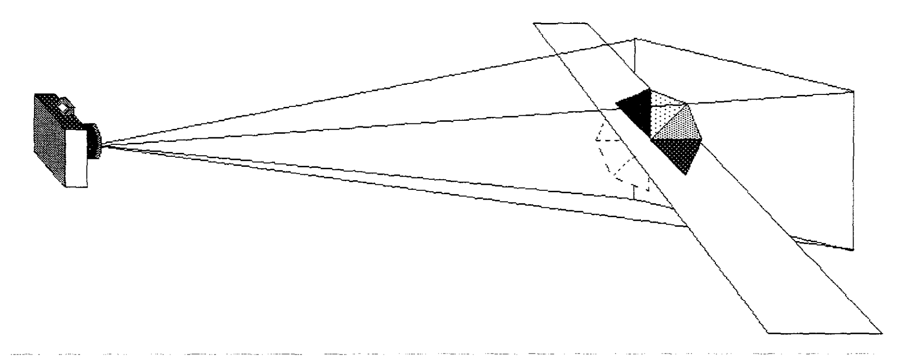
لإنشاء *viewport* عدد ٢:

```
glViewport (0, 0, sizex/2, sizey);
.
.
glViewport (sizex/2, 0, sizex/2, sizey);
```



سطوح قطع إضافية

بالإضافة إلى سطوح القطع الستة لفضاء الرؤية (يسار، يمين، أسفل، أعلى، قريب، بعيد) يمكنك تعريف حوالي 6 سطوح قطع إضافية كما في الشكل التالي:



وهذا يفيد في إزالة عناصر غير مرغوبة من المشهد.

الشكل العام لأمر تحديد سطح القطع :

```
void glClipPlane(GLenum plane, const GLdouble *equation);
```

يمثل الوسيط *plane* اسم مستوي القطع ، أما *equation* فتمثل معادلة مستوي القطع (تحدد المستوي المراد القطع وفقه) . معادلة مستوي القطع هي: $Ax+By+Cz+D=0$

يفعل الأمر السابق بـــ

```
glEnable(GL_CLIP_PLANEi);
```

ويلغى تفعيله بـــ

```
glEnable(GL_CLIP_PLANEi);
```

قيم *i* من *0* إلى *5* . وتحدد أحد مستويات القطع الستة المراد التعامل معها .

تطبيقات عملية

لتنفيذ التطبيقات التالية أنشئ تطبيقات فارغة بلغة *VC++* كما تعلمت ذلك في الفصل الأول ثم اكتب النص البرمجي المذكور في كل تطبيق .

تطبيق *1*

استخدام تحويلات *modeling*

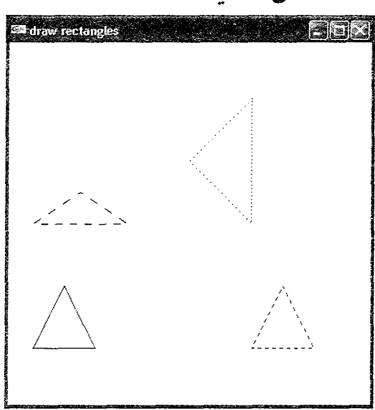
سنعمل على رسم أربعة مثلثات وتطبيق عدة تحويلات عليها وتطبيق نوع خط مختلف على كل مثلث. أنشئ تطبيقاً اسمه *application31* وأنشئ ضمنه الملف *a7.cpp* ثم اكتب النص البرمجي التالي ضمن الملف *a7.cpp* :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("draw rectangles");
 glutDisplayFunc(redraw);
 glMatrixMode(GL_PROJECTION);
 gluPerspective(45,1.0,10.0,200.0);
```

```
 glMatrixMode(GL_MODELVIEW);
 glutMainLoop();
 return 0;
}
static void redraw(void)
{
 float x1,x2,x3,y1,y2,y3;
#define draw_triangle(x1,y1,x2,y2,x3,y3)
glBegin(GL_LINE_LOOP);\
 glVertex2f(x1,y1);glVertex2f(x2,y2);glVertex2f(x3,y3);glEnd();
glClearColor(1.0,1.0,1.0,0.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glColor3f(0.0, 0.0, 0.0);
glEnable(GL_LINE_STIPPLE);
 //رسم المثلث اليساري السفلي
glLineStipple(1, 0xFFFF);
glLoadIdentity ();
glTranslatef (-15.0, 0.0, -70.0); /* viewing تحويل */
draw_triangle(-5.0,-10.0,-10.0,-20.0,0.0,-20.0);
 //رسم المثلث اليميني  السفلي
glLineStipple(1, 0xF0F0);
glLoadIdentity();
glTranslatef(10.0, 0.0, -70.0);
draw_triangle(5.0,-10.0,0.0,-20.0,10.0,-20.0);
 //رسم المثلث اليساري العلوي
glLineStipple(1, 0xF00F);
glLoadIdentity();
glTranslatef(-10.0, 0.0, -70.0);
glScalef(1.5, 0.5, 1.0);
draw_triangle(-5.,10.0,-10.0,0.0,0.0,0.0);
 //رسم المثلث اليميني العلوي
glLineStipple(1, 0x8888);
glLoadIdentity();
glTranslatef(20.0, 0.0, -70.0);
glRotatef (90.0, 0.0, 0.0, 1.0);
draw_triangle (10.0,20.0,0.0,10.0,20.0,10.0);
glDisable (GL_LINE_STIPPLE);
```

```
glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



تطبيق 2

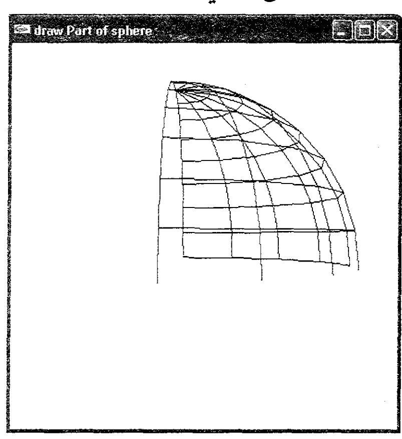
تطبيق سطوح قطع على كرة سلكية

سنعمل على رسم كرة سلكية وتطبيق سطحي قطع يستخدمان لقطع ¾ الكرة الأصلية. أنشئ تطبيقاً اسمه *application32* وأنشئ ضمنه الملف *a8.cpp* ثم اكتب النص البرمجي التالي ضمن الملف *a8.cpp* :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#include <gl\glaux.h>
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
 GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("draw Part of sphere");
 glutDisplayFunc(redraw);
```

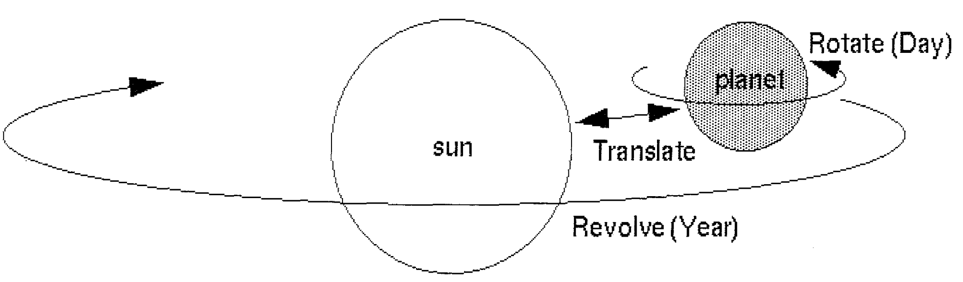
```
 glMatrixMode(GL_PROJECTION);
 gluPerspective(45,1.0,10.0,200.0);
 glMatrixMode(GL_MODELVIEW);
 glutMainLoop();
 return 0;
}
static void redraw(void)
{
 GLdouble eqn[4] = {0.0, 1.0, 0.0, 0.0};   /* y < 0 */
 GLdouble eqn2[4] = {1.0, 0.0, 0.0, 0.0};  /* x < 0 */
 glClearColor(1.0,1.0,1.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glTranslatef(0.0f,0.0f,-100.0f);
 glColor3f(0.0, 0.0, 0.0);
 glClipPlane (GL_CLIP_PLANE0, eqn);
 glEnable (GL_CLIP_PLANE0);
 glClipPlane (GL_CLIP_PLANE1, eqn2);
 glEnable (GL_CLIP_PLANE1);
 glRotatef (90.0, 1.0, 0.0, 0.0);
 auxWireSphere(30.0);
 glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



تطبيق 3
النظام الشمسي

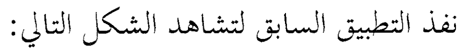
سنعمل على رسم كوكب وشمس ثم تدوير الكوكب حول محوره عكس عقارب الساعة وفي المدار حول الشمس مع عقارب الساعة . أنشئ تطبيقاً اسمه *application33* وأنشئ ضمنه الملف *a9.cpp* ثم اكتب النص البرمجي التالي ضمن الملف *a9.cpp* :

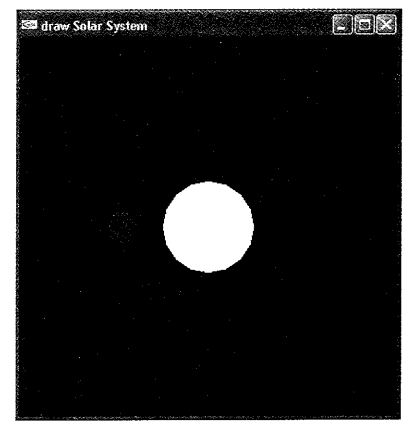



```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#include <gl\glaux.h>
static int year = 0, day = 0;
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
 GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("draw Solar System");
 glutDisplayFunc(redraw);
 glMatrixMode(GL_PROJECTION);


 gluPerspective(45,1.0,10.0,200.0);
 glMatrixMode(GL_MODELVIEW);
 glDepthFunc(GL_LEQUAL);
```

```
 glEnable(GL_DEPTH_TEST);
 glutMainLoop();
 return 0;
}
static void redraw(void)
{
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glTranslatef(0.0f,0.0f,-100.0f);
 glColor3f(1.0, 1.0, 1.0);
auxSolidSphere(10.0);            /* رسم الشمس */
glRotatef ((GLfloat) year, 0.0, 1.0, 0.0);
glTranslatef (20.0, 0.0, 0.0);
glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
glColor3f (1.0, 0.0, 0.0);
auxWireSphere(3.0);            /* رسم كوكب صغير */
day = (day + 10) % 360;/* دوران الكوكب حول نفسه */
year = (year - 1) % 360;/* دوران الكوكب حول الشمس */
 glutPostRedisplay();/* من أجل إعادة رسم الإطار لتطبيق تأثير الحركة */
 glutSwapBuffers();
}
```


نفذ التطبيق السابق لتشاهد الشكل التالي:



تطبيق 4

دوران الهرم والمكعب المرسومين في الفصل الثاني حول المحاور الإحداثية

سنعمل على تدوير المكعب حول المحور *x* مع عقارب الساعة والهرم حول المحور *y* عكس عقارب الساعة . أنشئ تطبيقاً اسمه *application34* وأنشئ ضمنه الملف *a10.cpp* ثم انسخ كامل النص البرمجي للملف *a6.cpp* التابع للتطبيق *4* ، ثم نفذ التعديلات التالية:

١. عرف المتحولات التي تتحكم بزاوية دوران المكعب والهرم بعد آخر تعليمة *include*:

```
#include <gl\glut.h>
static int rtri=0,rquad=0;// التعليمة الجديدة
```

٢. عدل تعليمة دوران الهرم الواقعة قبل تعليمة بدء رسم الهرم:

```
glRotatef(GLfloat(rtri),0.0f,1.0f,0.0f);// تعليمة جديدة
glBegin(GL_TRIANGLES);
```

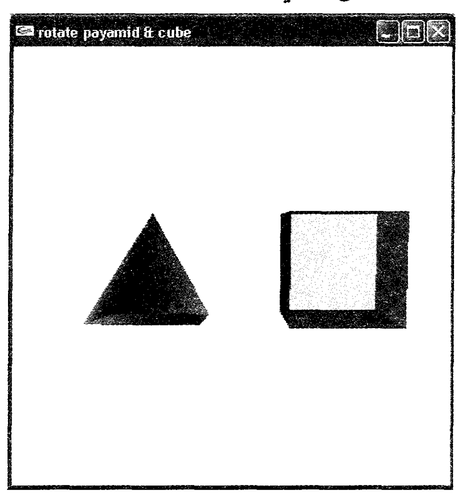
٣. عدل تعليمة دوران المكعب قبل تعليمة بدء رسم المكعب :

```
glRotatef(rquad,1.0f,0.0f,0.0f);// تعليمة جديدة
glBegin(GL_QUADS);
```

٤. أضف التعليمات التالية بعد الانتهاء من رسم المكعب:

```
glEnd();              // نهاية رسم المكعب //
rtri+= 1;      // زيادة زاوية دوران الهرم ١ درجة //
rquad-=2; // إنقاص  زاوية دوران المكعب ٢ درجة //
glutPostRedisplay();// تطبيق تأثير الحركة
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



الفصل الرابع

لوائح الإظهار

ستتعلم في هذا الفصل المواضيع التالية:

- مفهوم لائحة الإظهار
- إنشاء لائحة إظهار
- التعامل مع لوائح الإظهار
- تطبيقات عملية

مفهوم لائحة الإظهار

معظم إجراءات *OpenGL* يمكن استدعاؤها بشكل مباشر كما يمكن أن تخزّن في لائحة الإظهار*Display List* . لائحة الإظهار مجموعة ممن أوامر *OpenGL* التي يتم تخزينها من أجل تنفيذ لاحق .

ما الذي يخزن في لائحة الإظهار:

عند بناء لائحة الإظهار تخزن فقط قيم التعابير في اللائحة أي أن لائحة الإظهار تحتوي على استدعاءات لأوامر *OpenGL* المخزنة في اللائحة ، أما بقية المتحولات والاستدعاءات المكتوبة بلغات أخرى (لغة *C++* مثلاً) فيتم تقييمها ونسخها إلى لائحة الإظهار مع القيم التي تأخذها هذه المتحولات عندما تترجم اللائحة .

بعد أن تتم ترجمة اللائحة لا تتغير، أي يمكن حذف لائحة إظهار وإنشاء لائحة جديدة لكن لا يمكن تغيير لائحة إظهار موجودة .

<table>
<tr>
<td>

ملاحظة



</td>
<td>

ليست كل أوامر OpenGL قابلة للتخزين والتنفيذ من داخل لائحة الإظهار. وبشكل عام لا يمكن تخزين الأوامر التي تمرر وسائط كمراجع أو الأوامر التي تعيد قيمة ضمن لائحة إظهار . إذا استدعيت مثل هذه الأوامر عند بناء لائحة إظهار فستنفذ بشكل فوري ولن تخزن في لائحة الإظهار. نقدم فيما يلي الأوامر التي لا يمكن تخزينها في لائحة الإظهار (هناك بعض الأوامر الجديدة التي سنتعرف عليها لاحقاً).
glDeleteLists() , glIsEnabled() ,glFeedbackBuffer() , glIsList() , glFinish() , glPixelStore() , glFlush() , glReadPixels() , glGenLists() , glRenderMode() , glGet() , glSelectBuffer()*

</td>
</tr>
</table>

إنشاء لائحة إظهار

يتم إنشاء لائحة الإظهار بوضع النص البرمجي أو مجموعة الأوامر المطلوبة بين الاجرائين *glNewList* و *glEndList* . الشكل العام لإجراء بداية لائحة الإظهار:

void glNewList (GLuint list, GLenum mode);

list: رقم تعريف فريد للائحة الإظهار. إذا كانت قيمة *mode* هي *GL_COMPILE* تترجم الأوامر وتوضع في اللائحة دون تنفيذ . أما إذا كانت قيمتها *GL_COMPILE_AND_EXECUTE* فتنفذ الأوامر مباشرة وتخزن في لائحة الإظهار للاستخدام اللاحق .

الشكل العام لإجراء نهاية لائحة الإظهار:

void glEndList (void);

ملاحظة 	يجب إنهاء تشكيل لائحة قبل البدء بلائحة جديدة أي لا يمكن استدعاء الإجراء ()glNewList بين الاجرائين ( )glNewList و ()glEndList .يؤدي استدعاء ()glNewList قبل إغلاق اللائحة إلى توليد الخطأ GL_INVALID_OPERATION .

مثـــــال على استخدام لوائح الإظهار

يمكن رسم دائرة باستدعاء النص البرمجي التالي :

```
drawCircle()
{
GLint i;
GLfloat cosine, sine;
glBegin(GL_POLYGON);
 for(i=0;i<100;i++){
   cosine=cos(i*2*PI/100.0);
   sine=sin(i*2*PI/100.0);
   glVertex2f(cosine,sine);
```

```
}
glEnd();
}
```

عند استخدام هذا النص البرمجي في كل مرة يتم فيها تصيير الدائرة ، فإنه يجب حساب التوابع المثلثية وهذه طريقة غير فعالة .

أما عند استخدام لائحة الإظهار، فإنه يتم إجراء الحساب مرة واحدة عند ترجمة اللائحة ثم تخزن جميع القيم في اللائحة . لنكتب النص البرمجي السابق الخاص بالدائرة باستخدام لوائح الإظهار:

```
#define MY_CIRCLE_LIST1
buildCircle()
{
GLint i;
GLfloat cosine, sine;
glNewList(MY_CIRCLE_LIST1, GL_COMPILE);
glBegin(GL_POLYGON);
  for(i=0;i<100;i++){
    cosine=cos(i*2*PI/100.0);
    sine=sin(i*2*PI/100.0);
    glVertex2f(cosine,sine);
}
glEnd();
glEndList();
}
```

نستنتج مما سبق أن لوائح الإظهار في *OpenGL* مصممة للحصول على أداء أمثلي سريع وخاصة عبر الشبكات.

على سبيل المثال عند استخدام أمر بسيط مثل *glRotate()* في لائحة إظهار ، فسيبدي تحسناً ملحوظاً وذلك لأن الحسابات للحصول على مصفوفة الدوران ليست سهلة (تتضمن جذوراً تربيعية وتوابع مثلثية). أما باستخدام لائحة الإظهار ، فإنه يتم تخزين مصفوفة الدوران النهائية فقط وبالتالي ينفذ بسرعة أكبر عند استدعائه في لائحة إظهار.

تنفيذ لائحة إظهار

تُنفَّذ لائحة الإظهار باستدعاء التابع *glCallList( )* الذي يملك الشكل العام التالي:

Void glCallList(GLuint List);

ينفذ هذا التابع لائحة الإظهار المحددة بالوسيط *List* ،حيث تنفذ الأوامر حسب ترتيب ورودها في اللائحة .

ملاحظات 	١- إذا قمنا باستدعاء أوامر تحويل (مثلاً تحويل *Modeling*) ضمن لائحة إظهار فإن هذه الأوامر تؤثر على الرسومات اللاحقة . ٢- عندما تحتوي لائحة الإظهار على استدعاءات تغير في قيمة متحولات حالة ، فإنها تؤدي إلى تغيير قيمة متحولات الحالة في كل مرة تنفذ فيها اللائحة ، وتستمر هذه التغيرات بعد أن يتم تنفيذ لائحة الإظهار. ٣- يمكن إنشاء لائحة إظهار في تابع وتنفيذها في تابع مختلف . كما يتم إتلاف لائحة الإظهار إذا تم إتلاف سياق OpenGL التابعة له .

التعامل مع لوائح الإظهار

- يعّرف دليل لائحة إظهار اللائحة بشكل فريد .
- يجب الانتباه عند اختيار دليل لائحة إلى أننا قد نختار بشكل عرضي دليل قيد الاستخدام وبالتالي مسح لائحة إظهار مستخدمة .
- يمكن استخدام *glGenLists()* لتوليد دليل غير مستخدم و*glIsList()* لتحديد فيما إذا كان دليل معين قيد الاستخدام .الشكل العام لأمر *glGenLists()* :

GLuint glGenLists(GLsizei range);

يخصص *range* عدد من أدلة لائحة إظهار متسلسلة غير مستخدمة

GLboolean glIsList(GLuint list);

يعيد *True* إذا كان الدليل مستخدم كدليل لائحة إظهار و إلا يعيد *False* .

- الأمر *glDeleteLists()* يحذف مجموعة متتالية من لوائح الإظهار وبالتالي جعل هذه الأدلة متوفرة للاستخدام .

glDeleteLists(GLuint list, GLsizei range);

يحذف *range* لائحة إظهار بدءً من اللائحة ذات الدليل المحدد بالوسيط *list* .

- تكون لوائح الإظهار أكثر فعالية من الاستدعاء المباشر في الحالات التالية :
  - التبديل بين خرائط تراكيب متعددة texture maps .
  - التبديل بين الإعدادات المختلفة للإضاءة ونماذج الإضاءة ووسائط المواد و نماذج التهشير و ووسائط الضباب .

تطبيقات عملية

تطبيق 1

استخدام لوائح الإظهار *Display Lists*

سنعمل على رسم عدة مكعبات (١٥ مكعب) ضمن سلسلة هندسية تتألف من ٥ أسطر ، يوضع في السطر الأول مكعب واحد وفي السطر الثاني مكعبين وفي السطر الثالث ٣ مكعبات وهكذا . سنرسم المكعبات بدون الوجه العلوي *Top* وسنرسم كل سطر بلون . أنشئ تطبيقاً اسمه *application41* وأنشئ ضمنه الملف *a11.cpp* ، ولنشرح أجزاء التطبيق المضافة متمثلة في الخطوات التالية:

١. وضع التعريفات التالية:

```
GLuint box,top,xloop,yloop;
GLfloat xrot,yrot;
```

يتصرف المتحولان *box,top* كمؤشران لمكان تخزين اللائحة في الذاكرة *RAM* .أما المتحولان *xloop,yloop* فيحددان مكان توضع المكعبات على الشاشة . وأخيراً يستخدم المتحولان *xrot,yrot* لتدوير المكعبات حول *x,y* .

٢. تعريف مصفوفة ألوان :

بحيث تمثل كل قيمة فيها سطر من أسطر المكعبات (الألوان مرتبة حسب المصفوفة كما يلي: أحمر، برتقالي، أصفر،أخضر،أزرق) بالشكل التالي:

```
static GLfloat boxcol[5][3]=
{
{1.0f,0.0f,0.0f},{1.0f,0.5f,0.0f},{1.0f,1.0f,0.0f},{0.0f,1.0f,0.0f},{0
.0f,1.0f,1.0f}
};
```

٣. كتابة التابع *GLvoid BuildLists()* الذي يستخدم لبناء لائحة الإظهار وبناء مكعب ضمنها.

٤. استدعاء تابع بناء اللائحة ضمن التابع الرئيسي *main* :

```
BuildLists();
```

٥.كتابة التابع *redraw* الذي يرسم المكعبات باستدعاء لائحة الإظهار *.glCallList(box)*

يصبح النص البرمجي الكامل لهذا التطبيق (الملف *a11.cpp*):

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
GLuint box,top,xloop,yloop;
GLfloat xrot,yrot;
static GLfloat boxcol[5][3]=
{
 {1.0f,0.0f,0.0f},{1.0f,0.5f,0.0f},{1.0f,1.0f,0.0f},{0.0f,1.0f,0.
 0f},{0.0f,1.0f,1.0f}
};
GLvoid BuildLists();
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
 GLUT_DEPTH);
```

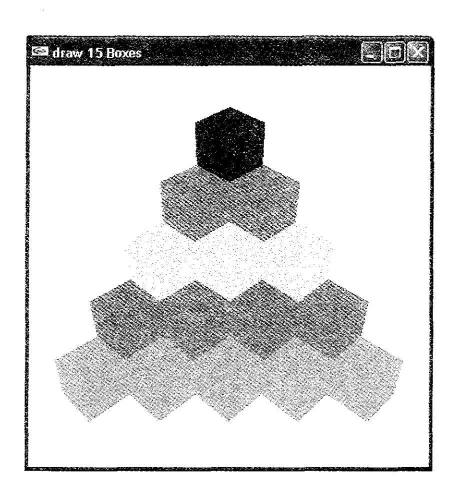
```
  glutInitWindowPosition(100,100);
  glutInitWindowSize(400,400);
  glutCreateWindow("draw 15 Boxes");
  glutDisplayFunc(redraw);
  BuildLists();
  glMatrixMode(GL_PROJECTION);
  gluPerspective(45,1.0,10.0,200.0);
  glMatrixMode(GL_MODELVIEW);
  glutMainLoop();
  return 0;
}
GLvoid BuildLists()
{
  box=glGenLists(1);// box توليد  دليل غير مستخدم للائحة ووضعه ضمن
  glNewList(box,GL_COMPILE); // البدء بلائحة المكعب
  glBegin(GL_QUADS);
              // الوجه السفلي للمكعب
//تحديد الناظم للوجه بحيث يكون من داخل الشاشة باتجاه عين الناظر//
              glNormal3f( 0.0f,-1.0f, 0.0f);
              glVertex3f(-1.0f, -1.0f, -1.0f);
              glVertex3f( 1.0f, -1.0f, -1.0f);
              glVertex3f( 1.0f, -1.0f,  1.0f);
              glVertex3f(-1.0f, -1.0f,  1.0f);
              // الوجه الأمامي للمكعب
              glNormal3f( 0.0f, 0.0f, 1.0f);
              glVertex3f(-1.0f, -1.0f,  1.0f);
              glVertex3f( 1.0f, -1.0f,  1.0f);
                  glVertex3f( 1.0f,  1.0f,  1.0f);
              glVertex3f(-1.0f,  1.0f,  1.0f);
              // الوجه الخلفي للمكعب
              glNormal3f( 0.0f, 0.0f,-1.0f);
              glVertex3f(-1.0f, -1.0f, -1.0f);
              glVertex3f(-1.0f,  1.0f, -1.0f);
              glVertex3f( 1.0f,  1.0f, -1.0f);
              glVertex3f( 1.0f, -1.0f, -1.0f);
              // الوجه اليميني للمكعب
```

```
                glNormal3f( 1.0f, 0.0f, 0.0f);
                glVertex3f( 1.0f, -1.0f, -1.0f);
                glVertex3f( 1.0f,  1.0f, -1.0f);
                glVertex3f( 1.0f,  1.0f,  1.0f);
                glVertex3f( 1.0f, -1.0f,  1.0f);
                // الوجه اليساري للمكعب
                glNormal3f(-1.0f, 0.0f, 0.0f);
                glVertex3f(-1.0f, -1.0f, -1.0f);
                glVertex3f(-1.0f, -1.0f,  1.0f);
                glVertex3f(-1.0f,  1.0f,  1.0f);
                glVertex3f(-1.0f,  1.0f, -1.0f);
        glEnd();
  glEndList();
}
static void redraw(void)
{
 glClearColor(1.0,1.0,1.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 for (yloop=1;yloop<6;yloop++)
 {
        for (xloop=0;xloop<yloop;xloop++)
        {
           glLoadIdentity();


           glTranslatef(1.4f+(float(xloop)*2.8f)-
           (float(yloop)*1.4f),((6.0f- loat(yloop))*2.4f)-
                   7.0f,-20.0f);
           glRotatef(45.0f-(2.0f*yloop)+xrot,1.0f,0.0f,0.0f);
           glRotatef(45.0f+yrot,0.0f,1.0f,0.0f);
           glColor3fv(boxcol[yloop-1]);
           glCallList(box);


        }
 }
 glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:





تطبيق 2

رسم عناصر هندسية باستخدام الرباعيات Quadrics

سنعمل على رسم عناصر هندسية (اسطوانة وقرص ودائرة) باستخدام الرباعيات *Quadrics* والمكتبة *GLU*. أنشئ تطبيقاً اسمه *application42* وأنشئ ضمنه الملف *a12.cpp* ، ولنشرح أجزاء التطبيق المضافة متمثلة في الخطوات التالية:

١. نضع في قسم تعريف المتحولات

```
GLUquadricObj *quadratic;
```

يعرف مؤشراً لمكان في الذاكرة يستخدم لتخزين العناصر الرباعية المراد رسمها.

٢. توليد المؤشر السابق وربطه مع مكان في الذاكرة يتم بوضع التعليمة التالية ضمن التابع الرئيسي *main* :

```
quadratic=gluNewQuadric();
```

٣. نرسم العناصر الهندسية التالية ضمن التابع *redraw*:

- *gluCylinder(quadratic,5.0,5.0,15.0,32,32);*

رسم اسطوانة نصف قطر القاعدة السفلية لها 5.0 و نصف قطر القاعدة العلوية لها 5.0 وارتفاعها 15.0 وعدد التقسيمات حول المحور z يساوي 32 وعدد التقسيمات على طول المحور z يساوي 32 .

- gluSphere(quadratic,6.0,32,32);

رسم كرة نصف قطرها 6.0 وعدد التقسيمات الطولية لها 32 وعدد التقسيمات العرضية لها 32 .

- gluDisk(quadratic,2.5,5.0,32,32);

رسم ديسك (قرص) نصف القطر الداخلي له 2.5 ونصف القطر الخارجي له 5.0 وعدد التقسيمات الطولية 32 وعدد التقسيمات العرضية (الحلقات) 32 .

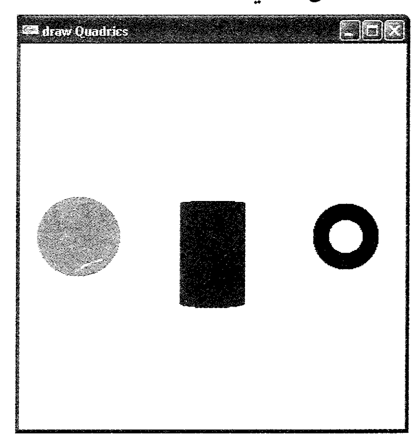
 	لمزيد من المعلومات حول العناصر الهندسية الرباعية السابقة ، راجع الملحق "ب".

يصبح النص البرمجي الكامل لهذا التطبيق (الملف a12.cpp):

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
GLUquadricObj *quadratic;
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
 GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("draw Quadrics");
 glutDisplayFunc(redraw);
 quadratic=gluNewQuadric();
```

```
  glMatrixMode(GL_PROJECTION);
  gluPerspective(45,1.0,10.0,200.0);
  glMatrixMode(GL_MODELVIEW);
  glutMainLoop();
  return 0;
}
static void redraw(void)
{
  glClearColor(1.0,1.0,1.0,0.0);
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  glTranslatef(0.0,5.0,-70.0);
  glColor3f(1.0, 0.0, 0.0);
  glRotatef(90,1.0,0.0,0.0);
  gluCylinder(quadratic,5.0,5.0,15.0,32,32);
  glLoadIdentity ();
  glTranslatef(-20.0,0.0,-70.0);
  glColor3f (0.0, 1.0, 0.0);
  gluSphere(quadratic,6.0,32,32);
  glLoadIdentity ();
  glTranslatef(20.0,0.0,-70.0);
  glColor3f (0.0, 0.0, 1.0);
  gluDisk(quadratic,2.5,5.0,32,32);
  glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



الفصل الخامس

الألوان

ستتعلم في هذا الفصل المواضيع التالية

- **تمييز الألوان**
- **ألوان الحاسب**
- **نمط *RGBA* ونمط *Color _index***
- **تخصيص نمط تظليل**
- **تطبيق عملي**

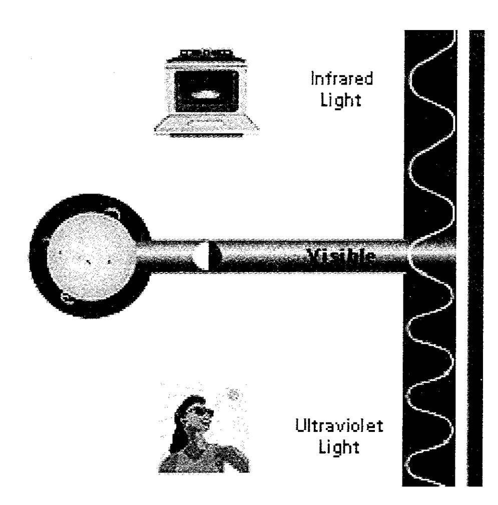
تمييز اللون

يتألف الضوء فيزيائياً من الفوتونات (ذرات صغيرة من الضوء) يتحرك كل منها ضمن مساره الخاص ، وكل منها يهتز حسب تردده الخاص (أو طول الموجة أو الطاقة الخاصة به) .يُوصف الفوتون بموقعه و اتجاهه وتردد (طول موجة) طاقته. مجال أطوال الموجات يتراوح بين 390nm (بنفسجي) و 720nm (أحمر) تغطي ألوان الطيف المرئي، و تشكِّل ألوان قوس قزح (بنفسجي، نيلي، أزرق، أخضر، أصفر، برتقالي، أحمر).

من ناحية أخرى فإن العين تميز العديد من الألوان غير الموجودة في قوس قزح مثل ألوان الأبيض و الأسود و البني والوردي. يتألف الضوء الأبيض المثالي من كميات متساوية من الأضواء من كل الترددات.

عين الإنسان تُميّز اللون عندما تُهيّج خلايا معينة في الشبكية تدعى المخاريط عن طريق اصطدام الفوتونات بها.

الأنواع الثلاثة المختلفة للمخاريط تستجيب بأفضل شكل إلى أطوال الموجات الثلاث المختلفة للضوء: أحد أنواع الخلايا المخروطية يستجيب بأفضل شكل للضوء الأحمر، والنوع الآخر للأخضر والآخر للأزرق. يبين الشكل التالي مجالات أطوال الموجات المرئية وفق البنفسجية وتحت الحمراء .



ألوان الحاسب

تُسبِّب التجهيزات *Hardware* في شاشة حاسب ملونة إرسال كميات مختلفة من الأضواء الأحمر والأخضر والأزرق من قِبَل النقاط الضوئية ، وتدعى القيم *R,G,B* وعادة تحزّم مع بعضها (وأحياناً مع قيمة رابعة تدعى *Alpha* أو *A*) وعندها تُدعى القيمة المحزومة *RGBA*.

معلومات الألوان لكل نقطة ضوئية يمكن أن تُخَزَّن إما بنمط *RGBA* حيث يتم الاحتفاظ بأربع قيم *R*، *G*، *B*، *A* لكل نقطة ضوئية، أو بنمط فهرس اللون حيث يُخَزَّن رقم واحد يُدعى فهرس اللون مع كل نقطة ضوئية.

كل فهرس يُشير إلى مدخل في جدول يعرّف مجموعة تفصيلية من القيم *R*، *G*، *B* . مثل هذا الجدول يُدعى خريطة اللون.

التجهيزات المختلفة للرسوميات تختلف بشكل كبير في كل من حجم مصفوفة النقاط الضوئية وعدد الألوان الممكن عرضها في كل نقطة ضوئية.

في نظام رسومي معين، كل نقطة ضوئية تملك نفس المساحة من الذاكرة لتخزين لونه وتسمى الذاكرة المحجوزة لكل النقاط الضوئية بالذاكرة المؤقتة للألوان.

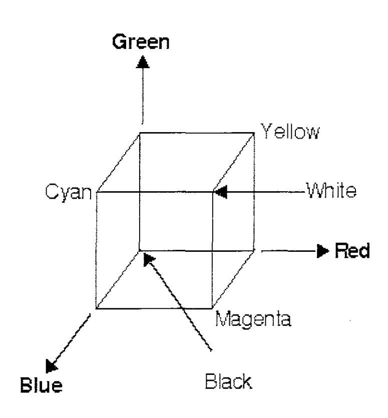
يُقاس حجم الذاكرة المؤقتة عادة بالبِت، فمثلاً الذاكرة المؤقتة ذات *8* بت يمكنها تخزين *256* لون مختلف لكل نقطة ضوئية، ويختلف حجم الذاكرة المؤقتة من آلة إلى أخرى.

يمكن أن تتراوح قيم *R*، *G*، *B* من *0.0* (لا قيمة) إلى *1.0* (كثافة كاملة). مثلاً *R=0.0, G=0.0, B=1.0* تمثل لون أزرق فاتح.

إذا كانت كل القيم = *0.0* ستكون النقطة الضوئية ذات لون أسود. أما إذا كانت كل القيم = *1.0* ستُعرض النقطة الضوئية بلون أبيض فاتح على الشاشة.

مزيج الأخضر والأزرق يُشكِّل السماوي، يتشكل البنفسجي من مزج الأحمر والأزرق، مزج الأحمر والأخضر يُشكِّل أصفر.يبين الشكل التالي مكعب الألوان حيث يمثل

اللون الأحمر *Red* المحور *x* واللون الأخضر *Green* المحور *y* واللون الأزرق *Blue* المحور *z* .
يمثل مركز الاحداثيات *(0,0,0)* اللون الأسود ، وطول ضلع المكعب يساوي *1* .



الأمر اللازم لإعطاء لون لعنصر معين يمكن أن يكون كالتالي:

```
glColor3f (1.0, 0.0, 0.0);  /* اللون المحدد هنا أحمر صافي */
glBegin (GL_POINTS);
  glVertex3fv (point_array);
glEnd ();
```

نمط *RGBA* ونمط *Color_Index*

تستطيع *OpenGL* العمل مع نمطين من الألوان هما نمط *RGBA* ونمط *Color_Index* . سيتم في كلا النمطين تخزين حجم محدد من المعطيات لكل نقطة ضوئية، ويتم تحديد هذا الحجم حسب عدد مستويات البتات *bitplanes* في الذاكرة المؤقتة للاطارات *framebuffer*، يحتوي مستوي البتات *bitplane* على بت واحد من البيانات لكل نقطة ضوئية. فإذا وُجد *8* مستويات بتات ملونة ، عندها سنستخدم *8 bits* للون من أجل كل نقطة ضوئية، وبالتالي سيكون لدينا $2^8 = 256$ قيمة مختلفة أو لون مختلف يتم تخزينه مع كل نقطة ضوئية.

تُقَسّم مستويات البتات على الألوان عادة عند تخزينها كـــ *R*، *G*، *B* (مثلاً نظام بـــ *24 bitplanes* سيخصِّص *8bit* للأحمر، *8bit* للأخضر، *8bit* للأزرق) .

لمعرفة عدد مستويات البتات الموجودة على نظامك من أجل كل من red ، green، blue، alpha أو قيم color_index ، استخدم الأمر glGetIntegerv() مع الوسائط GL_RED_BITS، GL_GREEN_BITS، GL_BLUE _BITS، GL_ALPHA _BITS، GL_INDEX _BITS.

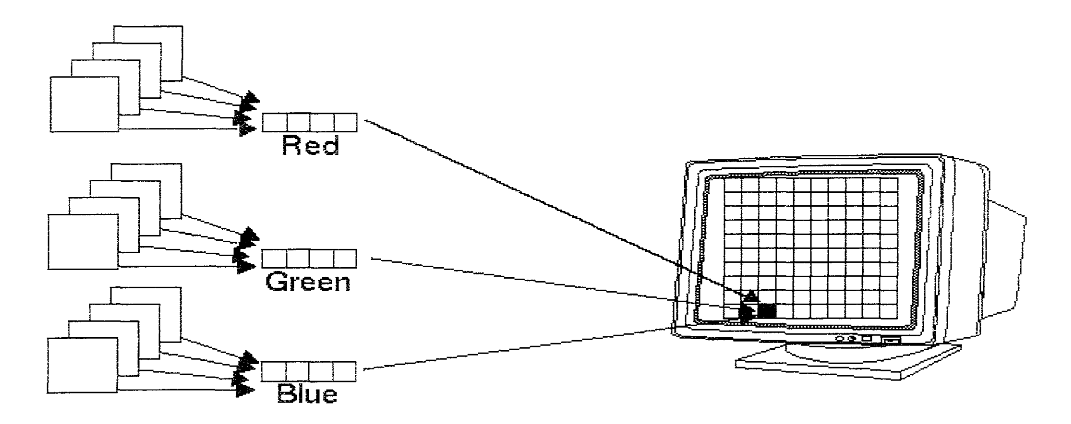
نمط العرض RGBA

تضع التجهيزات في نمط RGBA عدد محدد من مستويات البتات لكل من R، G، B، A .تخزّن قيم الـــ R، G، B، عادة كأعداد صحيحة بدلاً من أرقام ذات فاصلة عائمة. وبعدها يتم تحويلها إلى مجال البتات المتوفرة للتخزين والاسترجاع.



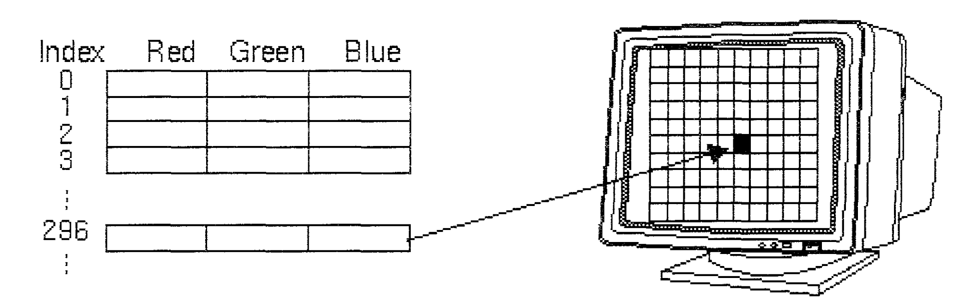
مثـــــال

إذا كان النظام يوفر 8bits للعنصر الأحمر، سنخزن أعداد صحيحة بين 0 و 255 أي 0، 1، 2، ... 255 في مستويات البتات، وستتعلق بقيم R التالية: 0.0 = 0/255، 1/255، 2/255 ، ... ، 1.0 = 255/255 . و بغض النظر عن عدد مستويات البتات، 0.0 تحدد أقل كثافة، 1.0 تحدد أعلى كثافة. قيمة alpha (A في RGBA) لا تملك تأثيراً مباشراً على اللون المعروض على الشاشة وتُستخدم في أشياء عديدة، بما فيها المزج والشفافية، وقد تؤثر على قيم R، G، B المكتوبة. عدد الألوان المتباينة الممكن عرضها مع كل نقطة ضوئية يعتمد على عدد مستويات البتات وإمكانية التجهيزات hardware في تفسير مستويات البتات هذه . هذا يعني أن آلة ذات 24 bitplanes لـــ RGB يمكنها عرض حتى 16.77 million لون مختلف. يبين الشكل التالي نظام ألوان يحوي 4 مستويات بتات لكل لون ، وبالتالي يمكن عرض عدد ألوان يساوي $4096 = 2^{12}$.

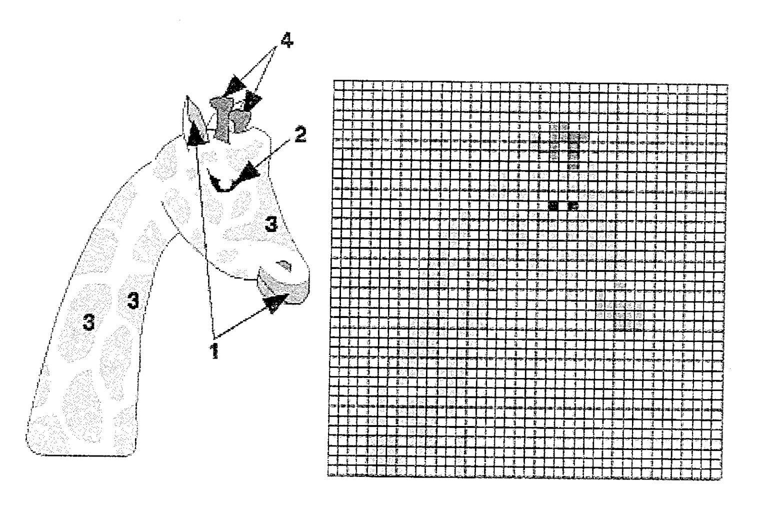


نمط العرض Color_Index

تستخدم *OpenGL* مع نمط العرض *Color_Index* خريطة ألوان تشبه استخدام صفيحة تحوي تقعرات لمزج الدهان (صفيحة يستخدمها الرسامون في الرسم). في الحاسب تزود فهارس يتم فيها مزج القيم الرئيسية للأحمر و الأخضر والأزرق.يبين الشكل التالي خريطة ألوان (تشبه الجدول) .



يخزّن الحاسب فهرس اللون في مستويات بتات لكل نقطة ضوئية . تعتبر بعد ذلك قيم مستويات البتات مراجع إلى خريطة الألوان،حيث تُرسم الشاشة بقيم الأحمر، الأخضر، الأزرق الموافقة من خريطة الألوان. تخزن هنا رقم الفهرس وليس قيمة اللون .يبين الشكل التالي تلوين زرافة بأرقام فهرس ، كل رقم يمثل لون معين .



في نظام Color_Index، عدد الألوان المتوفرة محدود بحجم خريطة الألوان وعدد مستويات البتات المتوفرة. يحدّد حجم خريطة الألوان بكمية ونوع التجهيزات hardware المخصصة له. الأحجام التقليدية تتراوح بين (2^8) 256 و (2^{12})4096 حيث أن حجم خريطة اللون هو 2 مرفوعة للقوة الممثلة بعدد مستويات البتات المتوفرة في نمط Color_Index.

إذا كان لدينا 2^n فهرس في خريطة الألوان و m bitplanes، فإن عدد المداخل المستخدمة هو الأصغر بين 2^nو 2^m.

في نمط RGBA يكون لون كل نقطة ضوئية مستقل عن النقاط الضوئية الأخرى. بينما في نمط فهرس الألوان كل نقطة ضوئية تملك نفس الفهرس المخزن ضمن bitplanes يتشارك على نفس الموقع في خريطة الألوان. إذا تغير محتوى أي مدخل في خريطة الألوان ستتغير ألوان جميع النقاط الضوئية ذات الفهرس المتغير.

الاختيار بين النمطين RGBA و Color_Index

تتم عملية الاختيار بين النمطين حسب التجهيزات الموجودة ، واحتياجات التطبيق. من أجل معظم الأنظمة، يمكن تمثيل مجموعة أكبر من الألوان باستخدام نظام RGBA. أيضاً

فإن نظام *RGBA* يؤمن مرونة أكبر من أجل التأثيرات المختلفة مثل التظليل و الإضاءة و تخطيط *texture* والضبابية.

يمكننا اختيار نمط فهرس الألوان في الحالات التالية:

١. إذا كنت تنقل تطبيقاً موجوداً يتعامل مع نمط الفهرس بشكل كبير، فإنه سيكون من الأسهل عدم التغيير إلى نمط *RGBA*.

٢. إذا كان لديك رقم صغير n من مستويات البتات وكنت بحاجة إلى أقل من 2^n لون مختلف، عليك استخدام نمط الفهرس.

٣. إذا كان لديك عدد محدود من مستويات البتات، من الممكن أن ينتج نمط الـــ *RGBA* ظلالاً سيئة بشكل واضح، وفي هذه الحالة يكون نمط الفهرسة أفضل إذا كان لديك متطلبات تظليل محدودة (على سبيل المثال فقط ظلال للرمادي).

٤. يمكن أن يكون نمط الفهرس مفيداً في خدع كثيرة، مثل رسم وتحريك الخرائط اللونية على طبقات.

بشكل عام استخدم نمط *RGBA* أفضل وأكثر استخداماً لأنه يعمل مع تخطيط التراكيب *textures* ويعمل بشكل أفضل مع الإضاءة و التظليل و الضبابية والمزج و والصقل.

الشكل العام لأمر الألوان باستخدام *RGBA*

بدون استخدام شعاع :

```
void glColor3{b s i f d ub us ui} (TYPEr, TYPEg, TYPEb);
void glColor4{b s i f d ub us ui} (TYPEr, TYPEg, TYPEb, TYPEa);
```

باستخدام شعاع :

```
void glColor3{b s i f d ub us ui}v (const TYPE*v);
void glColor4{b s i f d ub us ui}v (const TYPE*v);
```

يبين الجدول التالي تحويل قيم الألوان إلى أرقام ذات فاصلة عائمة:

البادئة	نوع البيانات	القيمة الصغرى	رقم الفاصلة العائمة الأصغري	القيمة العظمى	رقم الفاصلة العائمة الأعظمي
b	1-byte integer	-128	-1.0	127	1.0
s	2-byte integer	-32,768	-1.0	32,767	1.0
i	4-byte integer	-2,147,483,648	-1.0	2,147,483,647	1.0
ub	unsigned 1-byte integer	0	0.0	255	1.0
us	unsigned 2-byte integer	0	0.0	65,535	1.0
ui	unsigned 4-byte integer	0	0.0	4,294,967,295	1.0

من الجدول السابق نلاحظ أنه إذا استخدمنا نوع وسائط صحيحة (مثلاً *b*) فسيكون التعامل مع الألوان ضمن المجال من *-128* وحتى *127* . فمثلاً يعبر عن اللون الأخضر كما يلي:

```
glColor3b (-128,127,-128);
```

الشكل العام لأمر الألوان باستخدام فهرس الألوان:

```
void glIndex{sifd}(TYPE c);
void glIndex{sifd}v(const TYPE *c);
```

تخصيص نمط تظليل

يمكن رسم الخط أو المضلع المعبأ بلون واحد *(flat shading)* أو باستخدام عدة ألوان مختلفة متدرجة *(smooth shading)*.

نقوم بتحديد تقنية التظليل المرغوبة باستخدام الأمر:

```
void glShadeModel (GLenum mode);
```

بارمتر النمط *mode* يمكن أن يكون *GL_SMOOTH* (الخيار الافتراضي ويعطي تدرج لوني) أو *GL_FLAT* (لون واحد هو آخر لون).

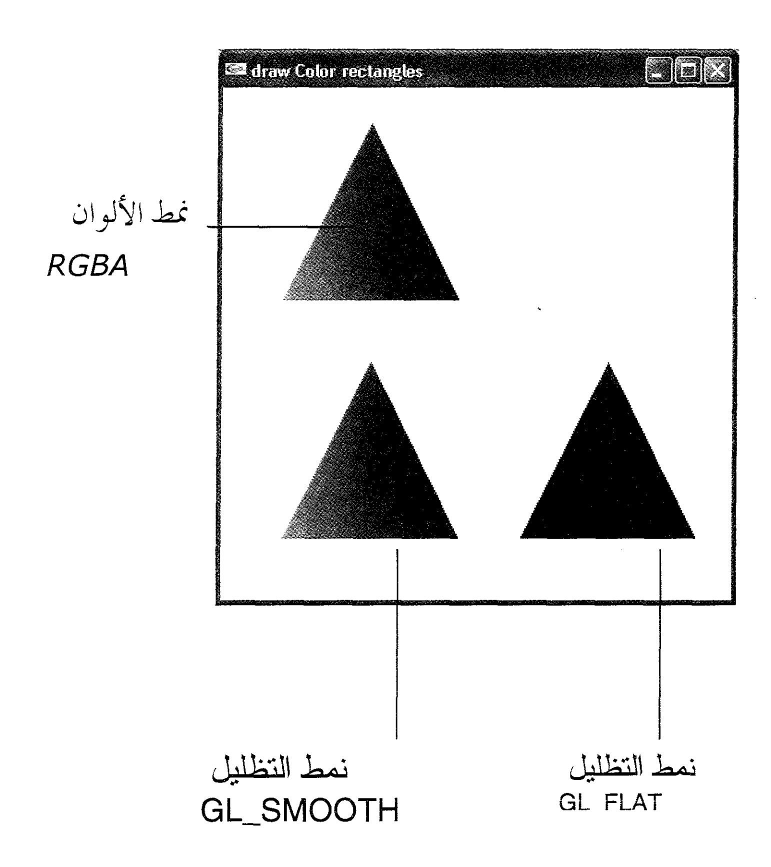
تطبيق عملي

سنعمل على رسم ثلاثة مثلثات، يطبق على أول مثلث من تلك المثلثات نمط التظليل *GL_SMOOTH* ، في حين يطبق على المثلث الثاني نمط التظليل *GL_FLAT* ويطبق على المثلث الثالث نمط الألوان *RGBA* . أنشئ تطبيقاً اسمه *application51* وأنشئ ضمنه الملف *a13.cpp* ، ثم اكتب النص البرمجي التالي ضمن الملف *a13.cpp* :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
GLUT_DEPTH);
glutInitWindowPosition(100,100);
glutInitWindowSize(400,400);
glutCreateWindow("draw Color rectangles");
glutDisplayFunc(redraw);
glMatrixMode(GL_PROJECTION);
gluPerspective(45,1.0,10.0,200.0);
glMatrixMode(GL_MODELVIEW);
glutMainLoop();
return 0;
}
static void redraw(void)
{
glClearColor(1.0,1.0,1.0,0.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glTranslatef(-12.0f,-12.0f,-70.0f);
/*smooth تدرج لوني*/
glShadeModel(GL_SMOOTH); //GL_SMOOTH تفعيل نمط التظليل
glBegin(GL_TRIANGLES);    // رسم مثلث
```

```
  glColor3f(1.0f,0.0f,0.0f); // أحمر
  glVertex3f( 0.0f, 10.0f, 0.0f);
  glColor3f(0.0f,1.0f,0.0f); // أخضر
  glVertex3f(-10.0f,-10.0f, 0.0f);
  glColor3f(0.0f,0.0f,1.0f); // أزرق
  glVertex3f( 10.0f,-10.0f, 0.0f);
glEnd();              // نهاية المثلث
/*flat لون واحد*/
glShadeModel(GL_FLAT);  //GL_FLAT تفعيل نمط التظليل
glLoadIdentity();
glTranslatef(15.0f,-12.0f,-70.0f);
glBegin(GL_TRIANGLES);  // رسم مثلث
  glColor3f(1.0f,0.0f,0.0f); // أحمر
  glVertex3f( 0.0f, 10.0f, 0.0f);
  glColor3f(0.0f,1.0f,0.0f); // أخضر
  glVertex3f(-10.0f,-10.0f, 0.0f);
  glColor3f(0.0f,0.0f,1.0f); // أزرق
  glVertex3f( 10.0f,-10.0f, 0.0f);
glEnd();
/* RGBA استخدام نمط الألوان */
glShadeModel(GL_SMOOTH);
glLoadIdentity();
glTranslatef(-12.0f,15.0f,-70.0f);
glBegin(GL_TRIANGLES);
  glColor3f(1.0f,0.0f,0.0f);
  glVertex3f( 0.0f, 10.0f, 0.0f);
  glColor3f(0.0f,1.0f,0.0f);
  glVertex3f(-10.0f,-10.0f, 0.0f);
  glColor3f(0.0f,0.0f,1.0f);
  glVertex3f( 10.0f,-10.0f, 0.0f);
glEnd();
glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



الفصل السادس

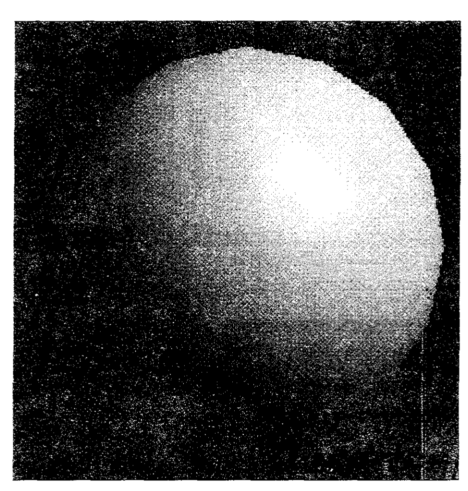
الإضاءة

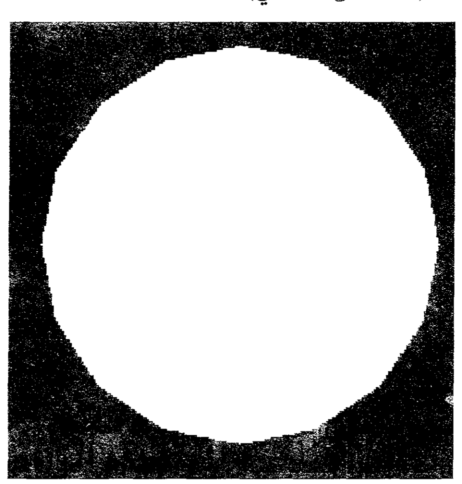
ستتعلم في هذا الفصل المواضيع التالية

- مفهوم الإضاءة
- ألوان المواد
- خصوات إضافة إضاءة الي المشهد
- تطبيقات عملية

مفهوم الإضاءة

معظم العناصر في الحقيقة لا يظهر شكلها ثلاثي الأبعاد إلا بتطبيق إضاءة عليها.يبين الشكل التالي كرة مع تطبيق إضاءة عليها (الشكل اليساري) ، وكرة بدون تطبيق إضاءة عليها (الشكل اليميني) .





تصدر الفوتونات من مصادر الضوء لتسقط على العناصر المضاءة. يُمتص بعض هذه الفوتونات ويُعكس بعضها من قبل السطح المضاء. حسب خصائص السطح يتم عكس الفوتونات في اتجاهات معينة أو في جميع الاتجاهات. تعامل *OpenGL* الإضاءة بتقسيمها إلى مكوناتها اللونية *RGB*. وبالتالي يتحدد لون مصادر الضوء حسب كمية *RGB* التي تصدرها. وتتحدد مادة السطح العاكس حسب نسبة المكونات *RGB* القادمة إلى السطح والتي تنعكس في اتجاهات معينة.

تأخذ *OpenGL* بعين الاعتبار تقسيم الإضاءة إلى أربعة مكونات مستقلة:

١. *Emitted*: أبسط أنواع الإضاءة. يصدر من العنصر ولا يتأثر بأي مصدر إضاءة آخر .

٢. *Ambient*: يأتي من جميع الاتجاهات. الإضاءة الخلفية في الغرفة هي مكون *Ambient*، يصل الضوء *Ambient* إلى العين بعد مروره على عدة أسطح.

٣. *Diffuse*: يأتي من مصدر وحيد وعندما يصطدم بالسطح ، ينتشر بشكل متساو في جميع الاتجاهات لذلك يظهر براقاً بشكل متساوٍ.

٤. *Specular*: يأتي من اتجاه معين ويرتطم بالسطح في اتجاه مفضل. يؤدي سقوط شعاع ليزري على مرآة عالية الجودة إلى إنتاج انعكاس *100% Specular*.

ألوان المواد

تعكس كرة حمراء جميع ألوان الإضاءة الحمراء القادمة إليها وتمتص ألوان الإضاءة الأخرى الخضراء والزرقاء. إذا وُضعت كرة حمراء أمام مصدر ضوء أخضر فستظهر سوداء (سيمتص الضوء الأخضر ولا يوجد مصدر ضوء أحمر فلن يكون هناك إضاءة) . تمتلك المواد ألوان *ambient* و*diffuse* و*specular* مختلفة.

يدمج انعكاس *ambient* للمادة مع المكون *ambient* لكل مصدر ضوء قادم وهكذا بالنسبة لـ *diffuse* و*specular*.

الضوء الواصل للعين يتجاهل جميع المؤثرات الأخرى وعلى افتراض أن مكون الضوء *(LR,LG,LB)* والمادة تمتلك المكونات *(MR,MG,MB)* فسيكون الضوء الواصل للعين *(LR*MR, LG*MG, LB*MB)* .

إذا كان لدينا مصدري ضوء *(R1,G1,B1)* و *(R2,G2,B2)* فالضوء الناتج *(R1+R2, G1+G2, B1+B2)*.

خطوات إضافة إضاءة إلى المشهد

١ . تعريف أشعة الناظم لكل نقطة من العناصر.

٢ . إنشاء وتعيين موقع مصدر ضوء واحد أو أكثر.

٣. إنشاء نموذج الإضاءة الذي يستخدم لتحديد مستوى الإضاءة *ambient* العامة والمنطقة المتأثرة بها من المسقط.

٤ . تعريف خصائص مواد العناصر في المشهد. ولنتناول هذه الخطوات بالتفصيل:

١. تعريف أشعة الناظم لكل نقطة من العناصر

تستخدم لتحديد اتجاه العنصر بالنسبة لمصدر الإضاءة. وهذا يحدد كمية الإضاءة الواصلة لكل نقطة من كل مصدر ضوء.

٢. إنشاء وتعيين موقع مصدر ضوء واحد أو أكثر

تمتلك مصادر الإضاءة عدة خصائص مثل اللون والموقع والاتجاه. الأمر المستخدم لتحديد جميع خصائص الإضاءة:

void glLight{if}[v](GLenum light, GLenum pname, TYPE param);

light يمكن أن تكون *GL_LIGHT0, GL_LIGHT1, ... , or GL_LIGHT7* وتحدد الضوء المنشئ.

Pname تحدد خصائص الضوء كما في الجدول التالي. *Param* يحدد قيمة لكل خاصية في *Pname*. ويمكن أن يكون مجموعة قيم.

اسم الوسيط	القيمة الافتراضية	المعنى
GL_AMBIENT	*(0.0, 0.0, 0.0, 1.0)*	القيمة اللونية *RGBA* لمكون *ambient* للضوء
GL_DIFFUSE	*(1.0, 1.0, 1.0, 1.0)*	القيمة اللونية *RGBA* لمكون *diffuse* للضوء
GL_SPECULAR	*(1.0, 1.0, 1.0, 1.0)*	القيمة اللونية *RGBA* لمكون *specular* للضوء
GL_POSITION	*(0.0, 0.0, 1.0, 0.0)*	احداثيات موقع الضوء *(x, y, z, w)*


أمثلة

```
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

يجب تفعيل الإضاءة باستخدام *glEnable(GL_LIGHTING)*،وتفعيل كل ضوء باستخدام اسم الإضاءة ، مثلاً *glEnable(GL_LIGHT0)*. إلغاء تفعيل الإضاءة يتم باستدعاء *glDisable()* مع الثوابت السابقة .

هناك نوعان لتحديد مكان توضع الضوء بالنسبة للعناصر:

النوع الأول *directional* (قيمة *w=0* بالنسبة لموقع الضوء) وله تأثير حزمة أشعة متوازية تصل أشعتها بشكل متتالي عبر الزمن مثل الشمس.

النوع الثاني *positional* (قيمة *w<>0* بالنسبة لموقع الضوء) تصدر الحزم الضوئية من مكان معين ويحدد لها مكان تأثير على عناصر المشهد كما في المصباح المكتبي.


مثـــــال

```
// directional(w=0) نوع الإضاءة
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

التحكم بموقع واتجاه مصادر الضوء

يعامل الضوء لتحديد موقعه واتجاهه تماماً كما يعامل أي عنصر هندسي. وبالتالي يتأثر الضوء بمصفوفة *ModelView* (تحويل *Projection* لا يؤثر في الإضاءة).


أمثلة

١. يبقى موقع الضوء ثابتاً لأن مصفوفة التحويل هي المصفوفة الواحدية

```
glViewport(0, 0, w, h);
glMatrixMode (GL_PROJECTION);
glLoadIdentity();
glMatrixMode (GL_MODELVIEW);
glLoadIdentity();
GLfloat light_position[] = { 1.0, 1.0, 1.0, 1.0 };
glLightfv(GL_LIGHT0, GL_POSITION, position);
```

٢. يتحرك الضوء حول عنصر معين مع الدوران بزاوية *spin*

```
GLfloat light_position[] = { 0.0, 0.0, 1.5, 1.0 };
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glTranslatef(0.0, 0.0, -5.0);
glPushMatrix();
glRotated((GLdouble) spin, 1.0, 0.0, 0.0);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glPopMatrix();
auxSolidTorus();
glPopMatrix();
```

٣. تحديد نموذج إضاءة

لتحديد نموذج إضاءة لدينا ثلاثة مكونات مع الأمر *glLightModel*()*.

أ- الضوء المحيط العام *Global Ambient Light*

يشارك كل مصدر ضوء بضوء *ambient* إلى المشهد. هناك أيضاً ضوء *ambient* آخر لا ينبع من أي مصدر ضوء (ضوء *ambient* عام)

مثـــال

```
GLfloat lmodel_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
```

ب- وجهة نظر (مكان رؤية) *ViewPoint* محلية *Local* أو غير محدودة *Infinite*

يتحدد موقع تأثير وجهة النظر حسب الحسابات المنفذة لتحديد المنطقة الأشد إضاءة الناتجة عن انعكاس المكون *Specular* للضوء. وبكلام أوضح ، تعتمد كثافة شدة الإضاءة في نقطة (رأس) معينة على شعاع الناظم للنقطة والاتجاه بين النقطة و مصدر الضوء والاتجاه بين النقطة ووجهة النظر *ViewPoint* .

في وجهة النظر غير المحدودة يكون الاتجاه بين وجهة النظر *ViewPoint* وأي نقطة في المشهد ثابتة . في وجهة النظر المحلية تكون النتائج أكثر واقعية لكن يجب حساب الاتجاه بين وجهة النظر وكل نقطة وبالتالي سينقص الأداء الكلي مع وجهة النظر هذه. وجهة النظر غير المحدودة هي الوجهة الافتراضية. إذا أردت تغييرها إلى محلية نفذ ما يلي:

```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
```

لتحويل وجهة النظر إلى غير محدودة مرة أخرى ، نفذ التعليمة السابقة مع الوسيط *GL_FALSE*

ج – إضاءة الوجهين

حسابات الإضاءة تنفذ لجميع المضلعات سواء كانت وجه أمامي أو خلفي. يفضل عدم إضاءة الوجه غير المرئي وأيضاً قد يضاء بشكل مختلف.



مثـــــال

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
```

في المثال السبق تتم إضاءة الوجهين . أما إذا أردنا إضاءة وجه واحد أمامي فقط ، فننفذ التعليمة السابقة مع الوسيط *GL_FALSE* .

٤. تعريف خصائص مواد العناصر في المشهد

سنعرف هنا خصائص المواد المتمثلة بألوان *ambient* و *diffuse* و *specular* واللمعان واللون الناتج عن أي ضوء منبعث.

الأمر المستخدم لتحديد خصائص مواد العنصر :

```
void glMaterial{if}[v](GLenum face, GLenum pname, TYPEparam);
```

face: يدل على الوجه من العنصر الذي ستطبق المادة عليه،وقيمه *GL_FRONT* أو *GL_BACK* أو *GL_FRONT_AND_BACK*

Pname: تعرف خاصية (وسيط)مادة محددة وتعين القيم لها بواسطة param .يبين الجدول التالي قيم الوسائط الممكنة لــ Pname.

اسم الوسيط	القيمة الافتراضية	المعنى
GL_AMBIENT	*(0.2, 0.2, 0.2, 1.0)*	لون *ambient* للمادة.
GL_DIFFUSE	*(0.8, 0.8, 0.8, 1.0)*	لون *diffuse* للمادة.
GL_AMBIENT_AND_DIFFUSE		لون *ambient* و *diffuse* للمادة.
GL_SPECULAR	*(0.0, 0.0, 0.0, 1.0)*	لون *specular* للمادة
GL_SHININESS	*0.0*	لمعان المنطقة *specular* للمادة
GL_EMISSION	*(0.0, 0.0, 0.0, 1.0)*	لون الانبعاث للمادة
GL_COLOR_INDEXES	*(0,1,1)*	الفهارس اللونية لــ *ambient, diffuse, and specular*


أمثلة

١.لون أزرق غامق لمناطق *ambient,diffuse* للوجه الأمامي والخلفي للعنصر.

```
GLfloat mat_amb_diff[] = { 0.1, 0.5, 0.8, 1.0 };
glMaterialfv(GL_FRONT_AND_BACK,    GL_AMBIENT_AND_DIFFUSE,
mat_amb_diff);
```

٢.لون أبيض فاتح لمنطقة *specular* مع لمعان منخفض للوجه الأمامي للعنصر .

```
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
```

```
GLfloat low_shininess[] = { 5.0 };
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
```

٣. **لون مخضر منبعث من الوجه الأمامي للعنصر .**

```
GLfloat mat_emission[] = {0.3, 0.2, 0.2, 0.0};
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
```

تطبيقات عملية

تطبيق 1

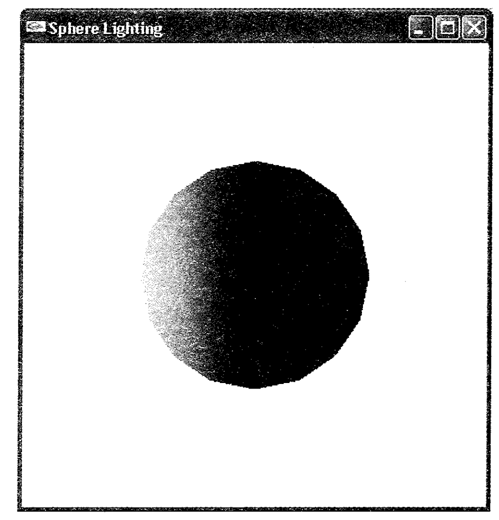
رسم كرة مضاءة

سنعمل على رسم كرة مضاءة مع تحديد لمعان وموقع الإضاءة . أنشئ تطبيقاً اسمه *application61* وأنشئ ضمنه الملف *a14.cpp* ، ثم اكتب النص البرمجي التالي ضمن الملف *a14.cpp* :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#include <gl\glaux.h>
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("Sphere Lighting");
 glutDisplayFunc(redraw);
 glMatrixMode(GL_PROJECTION);
 gluPerspective(45,1.0,10.0,200.0);
 glMatrixMode(GL_PROJECTION);
 glMatrixMode(GL_MODELVIEW);
```

```
  glutMainLoop();
  return 0;
}
static void redraw(void)
{
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_shininess[] = { 50.0 };
    GLfloat light_position[] = { -20.0, 0.0, 0.0, 0.0 };
    glClearColor(1.0,1.0,1.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0.0f,0.0f,-100.0);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    auxSolidSphere(20.0);
  glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



تطبيق 2

تحريك ضوء باستخدام تحويلات Modeling

سنعمل على رسم طارة *Torus* وتحريك وتدوير مصدر ضوء حولها. أنشئ تطبيقاً اسمه *application62* وأنشئ ضمنه الملف *a15.cpp* ، ثم اكتب النص البرمجي التالي ضمن الملف *a15.cpp* :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#include <gl\glaux.h>
static void redraw(void);
int main(int argc, char **argv);
static int spin = 0;//زاوية تدوير مصدر الإضاءة
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("move lighting");
 glutDisplayFunc(redraw);
 glMatrixMode(GL_PROJECTION);
 gluPerspective(45,1.0,10.0,200.0);
 glMatrixMode(GL_MODELVIEW);
 glEnable(GL_LIGHTING);//تفعيل الإضاءة
 glEnable(GL_LIGHT0);// light0 تفعيل الضوء
 glEnable(GL_COLOR_MATERIAL);//glColor يتأثر لون المادة باللون الحالي
 glutMainLoop();
 return 0;
}
static void redraw(void)
{
```

```
GLfloat position[] = { 0.0, 0.0, 1.5, 1.0 };
glClearColor(1.0,1.0,1.0,0.0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glTranslatef(0.0f,0.0f,-50.0);
glPushMatrix();
glRotated ((GLdouble) spin, 1.0, 0.0, 0.0);
glRotated (0.0, 1.0, 0.0, 0.0);
glLightfv (GL_LIGHT0, GL_POSITION, position);
glTranslated (0.0, 0.0, 15.0);
glDisable (GL_LIGHTING);
glColor3f (0.0, 1.0, 1.0);
auxWireCube (1.0);// ربط الضوء مع مكعب سلكي لرؤية حركته
glEnable (GL_LIGHTING);
glPopMatrix();
auxSolidTorus (2.75, 8.5);
glPopMatrix();
spin = (spin + 3) % 360;
glutPostRedisplay();
glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:





تطبيق 3

رسم مكعب مضاء بعدة مصادر إضاءة

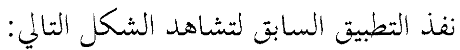
سنعمل على رسم مكعب وتطبيق إضاءة عليه بواسطة لوحة المفاتيح ، فعندما نضغط المفتاح *L* تطبق الإضاءة ، وعندما نضغط المفتاح *F* يلغى تطبيق الإضاءة . أنشئ تطبيقاً اسمه *application63* وأنشئ ضمنه الملف *a16.cpp* ، ثم اكتب النص البرمجي التالي ضمن الملف *a16.cpp* :

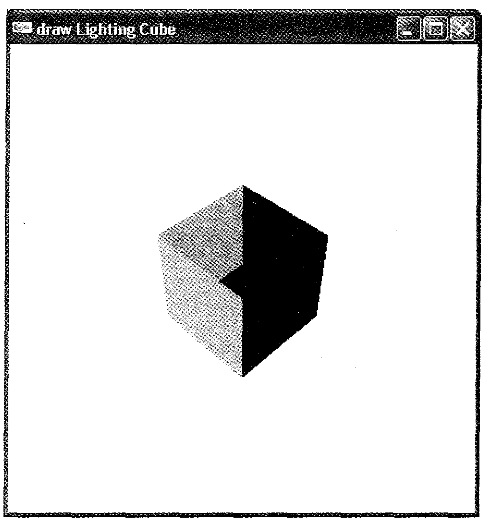
```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#include <gl\glaux.h>
GLfloat LightAmbient[]=    { 0.0f, 0.5f, 0.5f, 1.0f };
GLfloat LightDiffuse[]=    { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat LightPosition[]=   { 0.0f, 0.0f, 20.0f, 1.0f };
static void redraw(void);
int main(int argc, char **argv);
void keyboard ( unsigned char key, int x, int y );//لتعريف مفاتيح لوحة المفاتيح
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
 GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("draw Lighting rectangle");
 glutDisplayFunc(redraw);
 glMatrixMode(GL_PROJECTION);
 gluPerspective(45,1.0,10.0,200.0);
 glMatrixMode(GL_MODELVIEW);
glLightfv(GL_LIGHT1, GL_AMBIENT, LightAmbient); // Ambient إعداد الضوء
glLightfv(GL_LIGHT1, GL_DIFFUSE, LightDiffuse); //  Diffuse إعداد الضوء
 glLightfv(GL_LIGHT1, GL_POSITION,LightPosition);  // موقع الضوء
```

```
 glEnable(GL_LIGHT1); // LIGHT1 تفعيل الضوء
 glEnable(GL_COLOR_MATERIAL);
 glutMainLoop();
 return 0;
}
static void redraw(void)
{
 glClearColor(1.0,1.0,1.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glTranslatef(0.0f,0.0f,-100.0);
 glRotatef(45,1.0f,0.0f,0.0f);
 glRotatef(45,0.0f,1.0f,0.0f);
 glBegin(GL_QUADS);
        // الوجه الأمامي
        glColor3f(1.0,0.0,0.0);
        glNormal3f( 0.0f, 0.0f, 1.0f);
        glVertex3f(-10.0f, -10.0f,  10.0f);
        glVertex3f( 10.0f, -10.0f,  10.0f);
        glVertex3f( 10.0f,  10.0f,  10.0f);
        glVertex3f(-10.0f,  10.0f,  10.0f);
        // الوجه الخلفي
        glColor3f(1.0,1.0,0.0);
        glNormal3f( 0.0f, 0.0f,-1.0f);
        glVertex3f(-10.0f, -10.0f, -10.0f);
        glVertex3f(-10.0f,  10.0f, -10.0f);
        glVertex3f( 10.0f,  10.0f, -10.0f);
        glVertex3f( 10.0f, -10.0f, -10.0f);
        // الوجه العلوي
        glColor3f(0.0,1.0,0.0);
        glNormal3f( 0.0f, 1.0f, 0.0f);
        glVertex3f(-10.0f,  10.0f, -10.0f);
        glVertex3f(-10.0f,  10.0f,  10.0f);
        glVertex3f( 10.0f,  10.0f,  10.0f);
        glVertex3f( 10.0f,  10.0f, -10.0f);
        // الوجه السفلي
        glColor3f(1.0,0.0,1.0);
```

```
        glNormal3f( 0.0f,-1.0f, 0.0f);
        glVertex3f(-10.0f, -10.0f, -10.0f);
        glVertex3f( 10.0f, -10.0f, -10.0f);
        glVertex3f( 10.0f, -10.0f,  10.0f);
        glVertex3f(-10.0f, -10.0f,  10.0f);
        // الوجه الأيمن
        glColor3f(0.0,0.0,1.0);
        glNormal3f( 1.0f, 0.0f, 0.0f);
        glVertex3f( 10.0f, -10.0f, -10.0f);
        glVertex3f( 10.0f,  10.0f, -10.0f);
        glVertex3f( 10.0f,  10.0f,  10.0f);
        glVertex3f( 10.0f, -10.0f,  10.0f);
        // الوجه الأيسر
        glColor3f(0.0,1.0,1.0);
        glNormal3f(-1.0f, 0.0f, 0.0f);
        glVertex3f(-10.0f, -10.0f, -10.0f);
        glVertex3f(-10.0f, -10.0f,  10.0f);
        glVertex3f(-10.0f,  10.0f,  10.0f);
        glVertex3f(-10.0f,  10.0f, -10.0f);
  glEnd();
  glutKeyboardFunc    ( keyboard );
  glutPostRedisplay();
  glutSwapBuffers();
}
void keyboard ( unsigned char key, int x, int y )
{
  switch ( key ) {
    case 27:  /*  Escape key  */
      exit ( 0 );
      break;
    case 'L':
      glEnable(GL_LIGHTING);
      break;
    case 'F':
      glDisable(GL_LIGHTING);
      break;
    default:
      break;
        }
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:





تطبيق 4

تعديل على تطبيق1 في الفصل الرابع ويتمثل بإضافة إضاءة لسلسلة المكعبات الهندسية

سنعمل على إضافة الوجه العلوي للمكعبات السابقة باستخدام لوائح الإظهار ثم إضاءة تلك المكعبات لإعطائها صفة الواقعية. أنشئ تطبيقاً اسمه *application64* وأنشئ ضمنه الملف *a17.cpp* وذلك بنسخ الملف *a11* من التطبيق *application41* إلى التطبيق *application64* ثم عدل اسم الملف ليصبح *a17.cpp*، ثم نفذ التعديلات التالية على الملف المنسوخ *a17.cpp* :

١ . ضمن قسم التعريفات نكتب:

```
static GLfloat topcol[5][3]=
{
{.5f,0.0f,0.0f},{0.5f,0.25f,0.0f},{0.5f,0.5f,0.0f},{0.0f,0.5f,0.0f},{0
.0f,0.5f,0.5f}
};
```

تعريف مصفوفة ألوان بحيث تمثل كل قيمة فيها لون الوجه العلوي لسطر من أسطر المكعبات (الألوان مرتبة حسب المصفوفة كما يلي: أحمر غامق، برتقالي غامق، أصفر غامق ، أخضر غامق ،أزرق غامق).

٢ . ضمن التابع *BuildLists()* نضيف النص البرمجي التالي في نهايته قبل *glEnd()* لبناء الوجه العلوي :

```
// Top Face
glNormal3f( 0.0f, 1.0f, 0.0f);
glVertex3f(-1.0f,  1.0f, -1.0f);
glVertex3f(-1.0f,  1.0f,  1.0f);
glVertex3f( 1.0f,  1.0f,  1.0f);
glVertex3f( 1.0f,  1.0f, -1.0f);
glEnd();
```

٣ . ضمن التابع *main* نضيف النص البرمجي التالي لتفعيل الإضاءة :

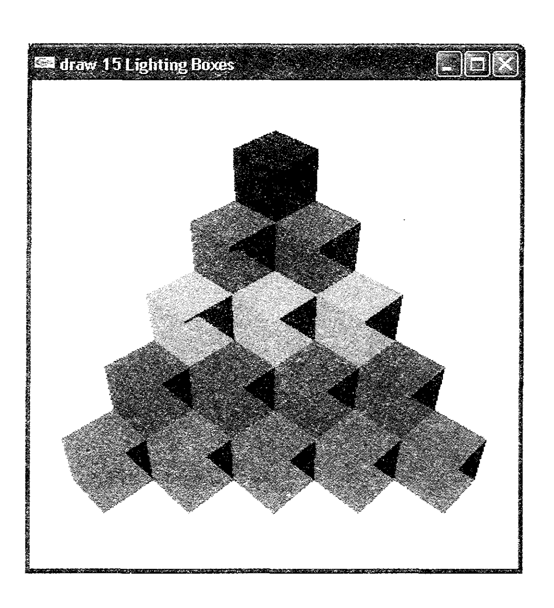
```
glEnable(GL_LIGHT0);
glEnable(GL_LIGHTING);
glEnable(GL_COLOR_MATERIAL);
```

٤ . يصبح التابع *redraw* :

```
static void redraw(void)
    {
    glClearColor(1.0,1.0,1.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    for (yloop=1;yloop<6;yloop++)
    {
       for (xloop=0;xloop<yloop;xloop++)
    {
             glLoadIdentity();  // Reset The View
             glTranslatef(1.4f+(float(xloop)*2.8f)-
             (float(yloop)*1.4f),((6.0f-float(yloop))*2.4f)-7.0f,-20.0f);
    glRotatef(45.0f-(2.0f*yloop)+xrot,1.0f,0.0f,0.0f);
    glRotatef(45.0f+yrot,0.0f,1.0f,0.0f);
    glColor3fv(boxcol[yloop-1]);
    glCallList(box);
    glColor3fv(topcol[yloop-1]);
    glCallList(top);
    }
    }
```

```
glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي



الفصل السابع

المزج و الصقل و الضباب

ستتعلم في هذا الفصل المواضيع التالية

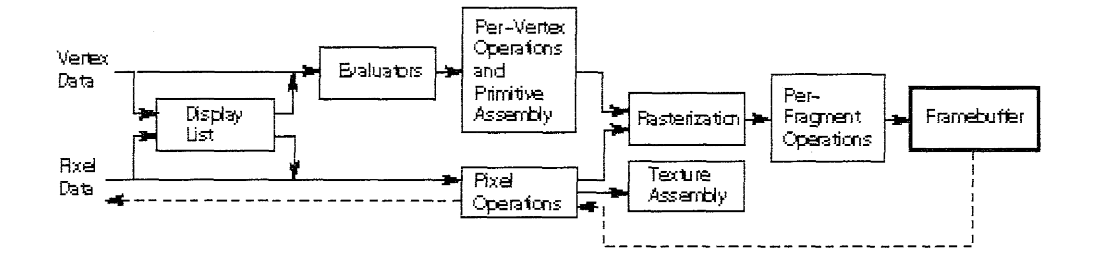
- مفهوم المزج
- المزج ثلاثي الأبعاد مع الذاكرة المؤقتة للعمق
- الصقل
- الضباب
- تطبيقات عملية

المزج Blending

عندما يتم تمكين المزج، تستخدم قيمة ألفا لمزج قيمة اللون الجديدة لنقطة ضوئية معطاة (مرحلة *Fragment*) مع لون النقطة الضوئية الموجودة مسبقاً والمخزنة في *framebuffer*. تتمثل أفضل طريقة لفهم المزج بالنظر إلى المركبة *RGB* على أنها تمثل اللون وقيمة المركبة ألفا تمثل درجة الشفافية .إذا كانت قيمة ألفا تساوي *0* فهذا يعني شفافية كاملة أما إذا كانت قيمة ألفا تساوي *1* فهذا يعني عدم وجود شفافية مطلقاً. يمكن وضع قيمة بين *0* و *1* للتدرج بالشفافية .

عند النظر إلى جسم عبر زجاج أخضر، اللون الناتج يكون لون الجسم بالإضافة إلى مزجه مع اللون الأخضر ونسبة المزج تختلف اعتماداً على خواص البث للزجاج فإذا كان الزجاج يرسل *80%* من الضوء الواقع عليه (له درجة كمد *20%*) فاللون الذي نراه هو مزيج *20%* من لون الزجاج و *80%* من لون الجسم.

تمر معلومات أي نقطة ضوئية تمثل صورة أو معلومات رأس يمثل شكلاً هندسياً بعدة مراحل قبل أن تخزن في الذاكرة المؤقتة للإطار framebuffer لإظهارها على الشاشة . تسمى المرحلة قبل الأخيرة Fragment وفيها يتم تطبيق الإكساء والمزج والصقل والضباب ، يبين الشكل التالي تلك المراحل:	**ملاحظة** 



عوامل المصدر والهدف

إذا كانت عوامل مزج المصدر والهدف هي: *(Sr,Sg,Sb,Sa)* للمصدر (القيم اللونية للنقاط الضوئية الجديدة) و *(Dr,Dg,Db,Da)* للهدف (القيم اللونية للنقاط الضوئية الموجودة مسبقاً) وقيم *RGBA* للمصدر والهدف تدل عليها اللاحقة *s* و *d* عندها فإن قيمة *RGBA* الممزوجة النهائية هي:

(RsSr+RdDr, GsSg+GdDg, BsSb+BdDb, AsSa+AdDa)

وكل مركبة تصبح داخل المجال *[0,1]*.

يمكن استخدام التابع *glBlendFunc()* لتحديد معاملات المصدر والهدف ولكن يجب تفعيل المزج أولاً عن طريق *glEnable(GL_BLEND)* ، ولإلغاء تفعيل المزج استخدم الأمر *glDisable(GL_BLEND)*.

الشكل العام لأمر المزج

```
void glBlendFunc(GLenum sfactor, GLenum dfactor)
```

يتحكم بكيفية مزج لون *fragment* مع اللون الموجود في *framebuffer*.

- *sfactor* يدل على كيفية حساب معامل مزج المصدر .
- *dfactor* يدل على كيفية حساب معامل مزج الهدف .

القيمة الافتراضية لـــ *sfactor* هي *GL_ONE* و *dfactor* هي *GL_ZERO* وتعطي نفس النتيجة في حال عدم تفعيل المزج.

القيم الممكنة لهذه المعاملات موجودة في الجدول التالي:

الثابت	المعاملات المرتبطة	حساب معامل المزج
GL_ZERO	المصدر أو الهدف	*(0, 0, 0, 0)*
GL_ONE	المصدر أو الهدف	*(1, 1, 1, 1)*

GL_DST_COLOR	المصدر	(Rd, Gd, Bd, Ad)
GL_SRC_COLOR	الهدف	(Rs, Gs, Bs, As)
GL_ONE_MINUS_DST_COLOR	المصدر	(1, 1, 1, 1)-(Rd, Gd, Bd, Ad)
GL_ONE_MINUS_SRC_COLOR	الهدف	(1, 1, 1, 1)-(Rs, Gs, Bs, As)
GL_SRC_ALPHA	المصدر أو الهدف	(As, As, As, As)
GL_ONE_MINUS_SRC_ALPH A	المصدر أو الهدف	(1, 1, 1, 1)-(As, As, As, As)
GL_DST_ALPHA	المصدر أو الهدف	(Ad, Ad, Ad, Ad)
GL_ONE_MINUS_DST_ALPH A	المصدر أو الهدف	(1, 1, 1, 1)-(Ad, Ad, Ad, Ad)
GL_SRC_ALPHA_SATURATE	المصدر	(f, f, f, 1); f=min(As, 1-Ad)

نماذج عن استخدام المزج

ليست كل التركيبات من معاملات المصدر والهدف لها معنى، أغلب التطبيقات تستخدم عدداً صغيراً من التركيبات.

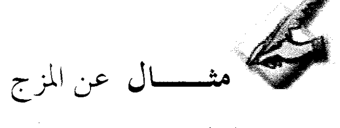


أمثلة

- إحدى الطرق لإنشاء رسمة مؤلف نصفها من صورة والنصف الآخر من صورة ثانية بقيمة مزج متساوية تتمثل بجعل معامل المصدر *GL_ONE* ورسم

الصورة الأولى ثم جعل معاملي المصدر والهدف GL_SRC_ALPHA ثم رسم الصورة الثانية مع قيمة ألفا *0.5*.

- إذا طُلب مزج *0.75* من الصورة الأولى مع *0.25* من الثانية: ارسم الصورة الأولى كما هو سابقاً ثم ارسم الصورة الثانية مع قيمة ألفا *0.25* لكن مع المعاملات *GL_SRC_ALPHA* للمصدر و*GL_ONE_MINUS_SRC_ALPHA* للهدف.
- لمزج ثلاث صور بالتساوي استخدم المعامل *GL_ONE* للهدف و *GL_SRC_ALPHA* للمصدر ، ارسم كل صورة مع قيمة ألفا مساوية لـــ *0.333333* . مع هذه التقنية كل صورة لها *1/3* من لمعانها الأصلي.



مثـــــال عن المزج

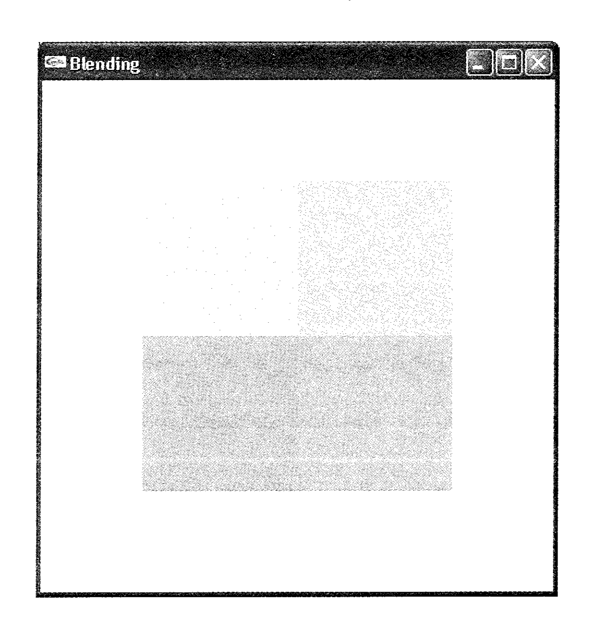
(المثال *example71* ضمن القرص الليزري المرفق مع الكتاب)

يرسم المثال التالي أربعة مستطيلات متداخلة ملونة. كل منها بقيمة ألفا *0.75* . المستطيلان السفلي اليساري (سماوي مع أصفر أصلي) والعلوي اليميني (أصفر مع سماوي أصلي) من الإطار يتم تغطيتهما مرتين. نعدل فقط التابع *redraw* ليصبح:

```
static void redraw(void)
{
glClearColor(1.0,1.0,1.0,0.0);//خلفية بيضاء بدون شفافية
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glTranslatef(-5.0f,-5.0f,-20.0);
glEnable(GL_BLEND);//تفعيل المزج
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);//اختيار معامل المزج للمصدر والهدف
glShadeModel(GL_FLAT);
glColor4f(1.0, 1.0, 0.0, 0.75);//0.75 صفر مع شفافية
glRectf(0.0, 0.0, 5.0, 10.0);
```

```
glColor4f(0.0, 1.0, 1.0, 0.75);//0.75 سماوي مع شفافية
glRectf(0.0, 0.0, 10.0, 5.0);
/* draw colored polygons in reverse order in upper right */
glColor4f (0.0, 1.0, 1.0, 0.75);
glRectf (5.0, 5.0, 10.0, 10.0);
glColor4f (1.0, 1.0, 0.0, 0.75);
glRectf (5.0, 5.0, 10.0, 10.0);
glutSwapBuffers();
}
```

نفذ المثال السابق لتشاهد الشكل التالي:



المزج ثلاثي الأبعاد مع الذاكرة المؤقتة للعمق

يؤثر ترتيب رسم المضلعات على نتيجة المزج. عند رسم الأجسام شبه الشفافة ثلاثية البعد، سنحصل على عدة أشكال اعتماداً على رسم المضلعات من الخلف إلى الأمام أو من الأمام إلى الخلف. كما يجب أن نأخذ بعين الاعتبار تأثير الذاكرة المؤقتة للعمق عند تحديد الترتيب الصحيح.نريد استخدام الذاكرة المؤقتة للعمق لإزالة أجزاء الأجسام المخفية التي تقع خلف الأجسام المعتمة. إذا وقع جسم معتم أمام جسم شبه شفاف أو جسم معتم آخر فإننا

نريد من الذاكرة المؤقتة للعمق أن تزيل الأجسام الأكثر بعداً. لكن إذا كان جسم شبه شفاف أقرب فإننا نريده أن يمزج مع الجسم المعتم . يمكن معرفة الترتيب الصحيح لرسم الأجسام إذا كان المشهد ثابتاً لكن هذا يصبح صعباً للغاية إذا كانت وجهة النظر *viewpoint* أو الجسم متحرك . الحل لهذه المشكلة يتمثل بتفعيل الذاكرة المؤقتة للعمق وجعلها *read-only* عند رسم الأجسام شبه الشفافة.

أولاً يجب رسم كل الأجسام المعتمة (غير الشفافة) مع الذاكرة المؤقتة للعمق في الحالة العادية. ثم نقوم بحفظ قيم العمق هذه بجعل الذاكرة المؤقتة للعمق *read-only*. تستخدم التعليمة *glDepthMask()* مع القيمة *GL_FALSE* لجعل الذاكرة المؤقتة للعمق *read-only* .



مثـــــال

(المثال *example72* ضمن القرص الليزري المرفق مع الكتاب)

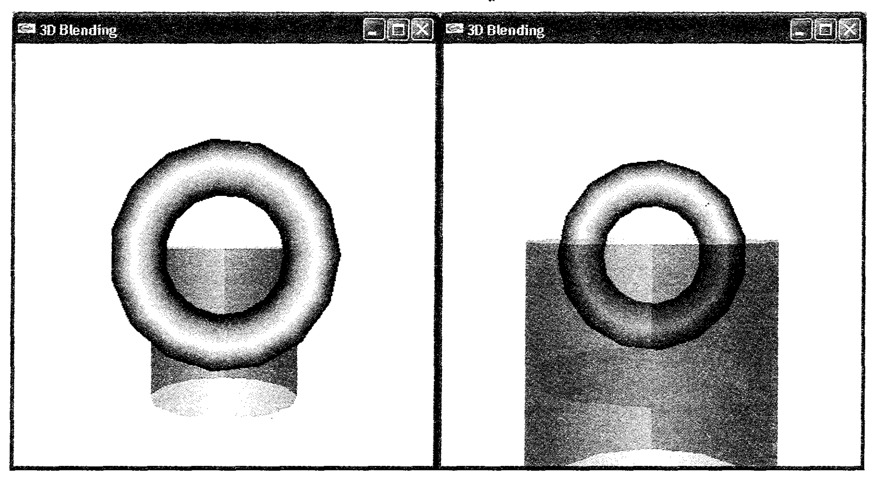
سنرسم في هذا المثال طارة *Tours* معتمة (كتيمة) وأسطوانة شفافة تقع خلف الطارة باتجاه العمق وننظر إلى العنصرين من الخارج (الطارة أمام الأسطوانة) بضغط المفتاح *o* من لوحة المفاتيح أو من الداخل (الأسطوانة امام الطارة) بضغط المفتاح *I* .

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#include <gl\glaux.h>
bool eyePosition;
static void redraw(void);
int main(int argc, char **argv);
void keyboard ( unsigned char key, int x, int y );
int main(int argc, char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
glutInitWindowPosition(100,100);
glutInitWindowSize(400,400);
```

```
glutCreateWindow("3D Blending");     glutDisplayFunc(redraw);
glMatrixMode(GL_PROJECTION);
gluPerspective(45,1.0,10.0,200.0);
glMatrixMode(GL_MODELVIEW);
glutMainLoop();
return 0;
}
static void redraw(void)
{
glClearColor(1.0,1.0,1.0,0.0);
GLfloat position[] = { 0.0, 0.0, 1.0, 1.0 };
GLfloat mat_torus[] = { 0.75, 0.75, 0.0, 1.0 };
GLfloat mat_cylinder[] = { 0.0, 0.75, 0.75, 0.35 };
GLfloat mat_ambient[] = { 0.0, 0.0, 0.0, 0.15 };
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 0.15 };
GLfloat mat_shininess[] = { 15.0 };
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glDepthFunc(GL_LEQUAL);
glEnable(GL_DEPTH_TEST);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glLightfv(GL_LIGHT0, GL_POSITION, position);
glPushMatrix();
if (eyePosition)
gluLookAt(0.0, 0.0, 9.0, 0.0, 0.0, 0.0, 0.0,1.0, 0.0);
else
gluLookAt(0.0, 0.0, -100.0, 0.0, 0.0, 0.0, 0.0,1.0, 0.0);
glPushMatrix();
glTranslatef(0.0, 0.0, -40.0);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_torus);
auxSolidTorus(2.75, 8.5);
glPopMatrix();
glEnable(GL_BLEND);
glDepthMask(GL_FALSE);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_cylinder);
glTranslatef(0.0, 0.0, -60.0);
```

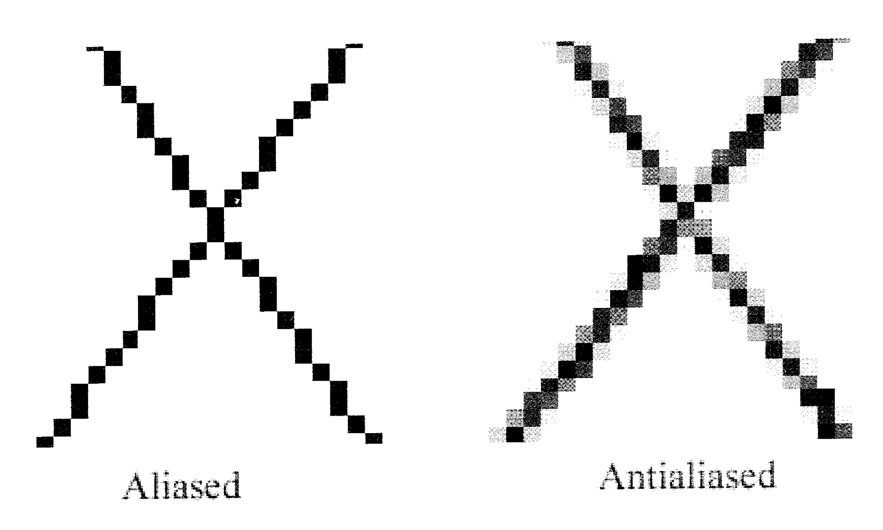
```
auxSolidCylinder(10.0, 20.0);
glDepthMask(GL_TRUE);
glDisable(GL_BLEND);
glPopMatrix();
glutKeyboardFunc( keyboard );
glutPostRedisplay();
glutSwapBuffers();
}
void keyboard ( unsigned char key, int x, int y )
    {
      switch ( key ) {
        case 27: /* Escape key */
          exit ( 0 );
          break;
        case 'o':
          eyePosition=true;
          break;
        case 'i':
          eyePosition=false;
          break;
        default:
          break;
                    }
    }
```

نفذ المثال السابق لتشاهد الشكل التالي:



الصقل Antialiasing

قد تلاحظ أحياناً في بعض صور *OpenGL* وجود خطوط أفقية أو شاقولية تظهر محززة (مشوهة). يظهر هذا التشويه لأن الخط يرسم من سلسلة من النقاط الضوئية المتقاربة. تدعى عملية التشويه بالتدرج *aliasing* ، وسيتم من خلال هذه الفقرة شرح تقنيات الصقل *Antialiasing* لإنقاص التدرج. يظهر الشكل التالي خطين متقاطعين بشكل متدرج و بشكل مصقول . و قد تم تكبير هذا الشكل لإظهار تأثير التشويه و الصقل.



يظهر الشكل السابق (القسم اليساري) خطين متقاطعين مشوهين وفيه تلاحظ مربعات ضوئية كبيرة وأخرى صغيرة. في الحقيقة ، عندما ننجز عملية صقل، فإن *OpenGL* تحسب قيمة التعبئة لكل جزء بالاعتماد على القسم من مربع النقطة الضوئية الذي سيغطيه الخط. يظهر الشكل السابق قيم التغطية من أجل الخط. في نمط *RGBA* ، تضرب *OpenGL* قيم ألفاً للجزء بقيمة التعبئة لها. تستطيع بعد ذلك استخدام القيمة الناتجة لألفا لمزج الجزء مع النقطة الضوئية المناسبة الموجودة مسبقاً ضمن الذاكرة المؤقتة للإطار *Framebuffer*. في نمط فهرسة اللون، تعين *OpenGL* على الأقل ٤ خانات عليا من فهرس اللون اعتماداً على تعبئة الجزء (*0000* من أجل عدم التعبئة و *1111* من أجل التعبئة الكاملة).

تستطيع استخدام الأمر *glHint()* لتطبيق بعض التحكمات على جودة و سرعة عمل الصور. الشكل العام لهذا الأمر:

```
void glHint(GLenum target, GLenum hint);
```

يشير الوسيط *target* إلى التصرف المراد التحكم به و قيمة مبينة في الجدول التالي :

الوسيط	المعنى
GL_POINT_SMOOTH_HINT, GL_LINE_SMOOTH_HINT, GL_POLYGON_SMOOTH_HINT	يحدد الجودة المرغوبة للنقاط والخطوط والمضلعات خلال عملية الصقل.
GL_FOG_HINT	يحدد فيما إذا كانت عمليات الضباب ستنفذ بالنقطة الضوئية *(GL_NICEST)* أو بالرأس *(GL_FASTEST)*.
GL_PERSPECTIVE_CORRECTION_HINT	يحدد الجودة المرغوبة للألوان وإحداثيات التراكيب .

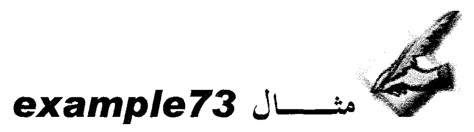
يمكن أن يأخذ الوسيط *hint* القيمة *GL_FASTEST* للإشارة إلى وجوب انتقاء الخيار الأكثر فاعلية، أو القيمة *GL_NICEST* للإشارة إلى الخيار الأكثر جودة، أو القيمة *GL_DONT_CARE* للإشارة إلى عدم وجود خيار معين.

صقل النقاط و الخطوط

تحتاج لصقل النقاط أو الخطوط إلى تفعيل الصقل باستخدام *glEnable()* مع الثابت *GL_POINT_SMOOTH* أو *GL_LINE_SMOOTH* . قد ترغب أيضاً بتأمين تأثير جودة باستخدام *glHint()* . سنتعلم صقل النقاط و الخطوط في نمطي الألوان.

نمط RGBA

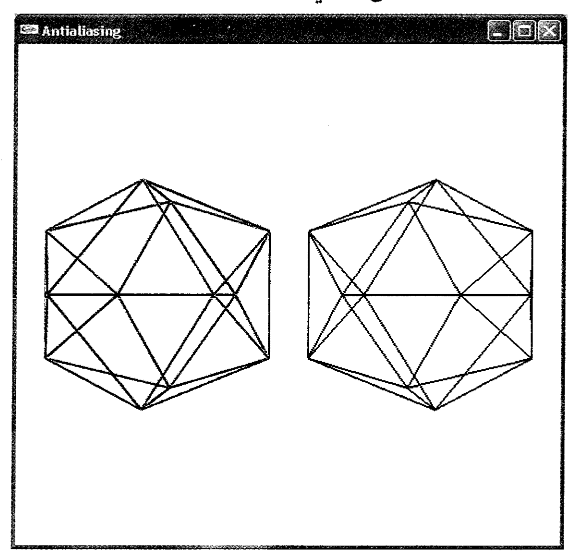
تحتاج هنا إلى تفعيل المزج. معامل المزج الأكثر استخداماً هنا هو GL_SRC_ALPHA من أجل المصدر و GL_ONE_MINUS_SRC_ALPHA من أجل الهدف . و بشكل بديل، تستطيع استخدام GL_ONE من أجل معامل الهدف لجعل الخطوط أقل لمعاناً عند التقاطعات. أنت الآن جاهز لرسم النقاط أو الخطوط التي تريد صقلها. يمكن ملاحظة تأثير الصقل بوضوح عند استخدام قيم عالية لألفا. يوضح المثال التالي عملية تهيئة الأنماط الضرورية للصقل ثم رسم العنصر الهندسي Icosahedron بشكل سلكي . لاحظ أن نمط الذاكرة المؤقتة للعمق غير مفعلة.



```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#include <gl\glaux.h>
static void redraw(void);
int main(int argc, char **argv);
void myinit(void)
{
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glHint(GL_LINE_SMOOTH_HINT, GL_DONT_CARE);
glLineWidth(2);
glShadeModel(GL_FLAT);
glDepthFunc(GL_LEQUAL);
glEnable(GL_DEPTH_TEST);
}
int main(int argc, char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
glutInitWindowPosition(100,100);
glutInitWindowSize(500,500);
```

```
glutCreateWindow("Antialiasing");
myinit();
glutDisplayFunc(redraw);
glMatrixMode(GL_PROJECTION);
gluPerspective(45,1.0,10.0,200.0);
glMatrixMode(GL_MODELVIEW);
glutMainLoop();
return 0;
}
static void redraw(void)
{
glClearColor(1.0,1.0,1.0,0.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glTranslatef(-20.0f,0.0f,-100.0f);
glColor3f(0.0, 0.0, 0.0);
glEnable(GL_LINE_SMOOTH);
auxWireIcosahedron(20.0);
glDisable(GL_LINE_SMOOTH);
glTranslatef(40.0f,0.0f,0.0f);
auxWireIcosahedron(20.0);
glutSwapBuffers();
}
```

نفذ المثال السابق لتشاهد الشكل التالي:



نمط فهرس اللون

يتمثل الجزء الأهم المتعلق بالصقل في نمط فهرس اللون بتحميل و استخدام خريطة اللون. و بما أن الخانات الأربعة الأخيرة من فهرس اللون تشير إلى قيمة التعبئة، فأنت بحاجة إلى تحميل *16* فهرس متتالية بلون تأثير مأخوذ من لون الخلفية بحيث يكون الأقرب للون العنصر (منطقة التأثير بقيمة فهرس من مضاعفات *16*) بعد ذلك يجب عليك مسح الذاكرة المؤقتة للون بأول لون من الألوان الستة عشر ضمن منطقة التأثير و رسم نقاطك أو خطوطك باستخدام ألوان من منطقة التأثير.

صقل المضلعات

تشبه عملية صقل حواف المضلعات الممتلئة كثيراً عملية صقل النقاط و الخطوط. عندما تمتلك مضلعات مختلفة حواف متداخلة، فستحتاج إلى القيمة اللونية المناسبة. إذا رسمت مضلعاتك كنقاط أو كحواف فقط و ذلك بتمرير الثابت *GL_POINT* أو *GL_LINE* إلى الأمر *glPolygonMode()* ، فبإمكانك تطبيق صقل النقاط أو الخطوط. نشرح فيما يلي حالة صقل مضلعات ممتلئة (أي نمط المضلع *GL_FILL*).

بشكل نظري، تستطيع صقل المضلعات في نمط *RGBA* و نمط فهرس اللون. و بشكل عام تأثير الصقل على مناطق تقاطع المضلعات أكبر منه في حالة التأثير على النقاط و الخطوط. إذا كان لديك عدة مضلعات فأنت بحاجة لتحديد ترتيبها من الأمام للخلف، ثم استخدام *glBlendFunc()* مع القيمة *GL_SRC_ALPHA_SATURATE* من أجل المصدر و *GL_ONE* من أجل الهدف.

تحتاج لصقل مضلعات في نمط *RGBA* إلى استخدام قيمة الشفافية *A* لتمثيل قيم التعبئة لحواف المضلعات. أنت بحاجة لتفعيل صقل المضلعات و ذلك بتمرير القيمة *GL_POLYGON_SMOOTH* إلى الأمر *glEnable()*. سيؤدي ذلك إلى إسناد قيم شفافية جزئية للنقاط الضوئية المتوضعة على حواف المضلع و ذلك اعتماداً على التعبئة. يمكنك أيضاً تزويد القيمة *GL_POLYGON_SMOOTH_HINT* إلى الأم *glHint()* .

تحتاج الآن إلى مزج الحواف المتداخلة بشكل مناسب. أولاً أوقف تشغيل الذاكرة المؤقتة للعمق ليمكنك التحكم بكيفية رسم النقاط الضوئية المتداخلة. ثم عين القيمة *GL_SRC_ALPHA_SATURATE* للمصدر و *GL_ONE* للهدف. بهذا التابع الخاص يكون اللون النهائي ناتج عن جمع لون المصدر مع لون المصدر النسبي. المعامل النسبي عبارة عن القيمة الأصغر بين قيمة ألفا للمصدر القادم أو واحد مطروحاً منه قيمة ألفا للهدف. هذا يعني أنه من أجل نقطة ضوئية بقيمة شفافية ألفا عالية، فستكون النقاط الضوئية القادمة ذات تأثير أقل في اللون الأخير لأن ناتج طرح شفافية الهدف من *1* ستقترب من الصفر. أخيراً أنت بحاجة لترتيب جميع المضلعات في مشهدك من الأمام للخلف قبل رسمها.

يظهر المثال التالي كيفية صقل مضلعات ممتلئة. تستطيع تفعيل أو تعطيل الصقل بالنقر على زر الفأرة الأيسر. لاحظ أنه تم حذف الأوجه الخلفية للمضلعات و تم مسح قيمة الشفافية إلى القيمة صفر ضمن الذاكرة المؤقتة للون قبل إنجاز أي عملية رسم.

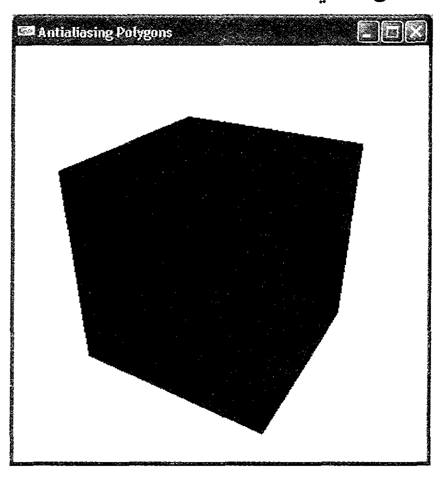
مثـــــال *example74*

يتم في هذا المثال صقل أوجه مكعب ، فعندما تنقر زر الفأرة الأيسر يطبق الصقل ، وعندما تحرر الزر يلغى تأثير الصقل.

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#include <gl\glaux.h>
static void redraw(void);
void myinit(void)
{
   glCullFace (GL_BACK);
   glEnable (GL_CULL_FACE);
   glDisable(GL_POLYGON_SMOOTH);
   glClearColor (1.0, 1.0, 1.0, 0.0);
}
void mouse ( int button, int state, int x, int y )
```

```
{
        if ( button == GLUT_LEFT_BUTTON && state ==
            GLUT_DOWN )
                {
            glEnable (GL_POLYGON_SMOOTH);
 }
if ( button == GLUT_LEFT_BUTTON && state == GLUT_UP )
                {
        glDisable (GL_POLYGON_SMOOTH);
            }
}
int main(int argc, char **argv)
{
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
 GLUT_DEPTH);
  glutInitWindowPosition(100,100);
  glutInitWindowSize(400,400);
  glutCreateWindow("Antialiasing Polygons");
  myinit();
  glutMouseFunc(mouse);
  glutDisplayFunc(redraw);
  glMatrixMode(GL_PROJECTION);
  gluPerspective(45,1.0,10.0,200.0);
  glMatrixMode(GL_MODELVIEW);
  glutMainLoop();
  return 0;
}
static void redraw(void)
{
glLoadIdentity();
glColor3f(0.0,0.0,1.0);
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glTranslatef (0.0, 0.0, -50.0);
glRotatef (30.0, 1.0, 0.0, 0.0);
glRotatef (60.0, 0.0, 1.0, 0.0);
auxSolidCube (20.0);
glutPostRedisplay();
glutSwapBuffers();
}
```

نفذ المثال السابق لتشاهد الشكل التالي:



الضباب Fog

قد تبدو أحياناً صور الحاسب ذات حواف غير واقعية. الصقل يجعل العنصر يظهر بشكل أكثر واقعية و ذلك بتنعيم جوانبه. تستطيع بالإضافة إلى ذلك جعل كامل الصورة تبدو أقرب إلى الواقع بإضافة الضباب، الذي يجعل العناصر تتلاشى بشكل تدريجي مع ازدياد المسافة (البعد عن عين الناظر). الضباب كمصطلح عام يصف صيغاً تشبه تأثيرات الغلاف الجوي، إذ يمكن استخدامه لمحاكاة الغيوم و الغباشة (الضباب) و الدخان و التلوث.

عند تفعيل الضباب، فإن العناصر الأبعد عن وجهة النظر تبدأ بالتلاشي لتتحول إلى لون الضباب. تستطيع التحكم بكثافة الضباب، و ذلك بتحديد نسبة تبدأ العناصر عندها بالتلاشي مع ازدياد المسافة. الضباب متاح في نمطي *RGBA* و فهرس اللون . يطبق الضباب بعد إنجاز مصفوفة التحويلات و الإضاءة و الإكساء، فهو يتأثر بالتحولات و الإضاءة و عناصر الإكساء.

استخدام الضباب

استخدام الضباب عملية سهلة. تستطيع تفعيله بتمرير القيمة *GL_FOG* إلى الأمر *glEnable()* و يمكنك تحديد لون الضباب و المعادلة التي تضبط كثافته بواسطة الأمر *glFog*()* . يمكنك تزويد القيمة *GL_FOG_HINT* إلى الأمر *glHint()* .

معادلات الضباب

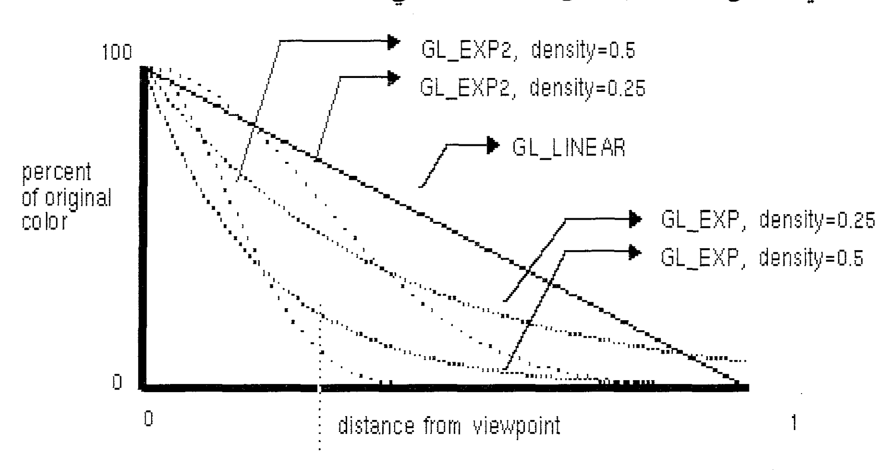
يمزج الضباب لونه مع اللون القادم من لون مرحلة *Fragment* باستخدام معامل مزج الضباب. هذا المعامل *f* يحسب باستخدام إحدى المعادلات الثلاث التالية ثم يحول إلى المجال *[0,1]* :

$$f = e^{-(density \cdot z)} \qquad \text{(GL_EXP)}$$

$$f = e^{-(density \cdot z)^2} \qquad \text{(GL_EXP2)}$$

$$f = \frac{end - z}{end - start} \qquad \text{(GL_LINEAR)}$$

تمثل *z* المسافة بالإحداثيات العينية بين وجهة النظر و مركز *fragment* . تحدد القيمة *density* و *start* و *end* باستخدام الأمر *glFog*()* . يبين الشكل التالي تطبيق معادلات الضباب من أجل وسائط مختلفة. يمثل المحور الأفقي البعد عن وجهة النظر ، أما المحور الشاقولي فيمثل نسبة مئوية من اللون الأساسي :



الشكل العام لأمر الضباب:

void glFog{if}{v}(GLenum pname, TYPE param);

إذا كانت قيمة *pname* تساوي *GL_FOG_MODE*، فستكون قيمة *param* إما *GL_EXP* (الخيار الافتراضي) أو *GL_EXP2* أو *GL_LINEAR* ، بذلك تتحدد نوع

معادلة الضباب. إذا كانت قيمة *pname* تساوي *GL_FOG_DENSITY* أو *GL_FOG_START* أو *GL_FOG_END* ، فستكون قيمة *param* عبارة عن قيمة (مصفوفة)*density* للكثافة أو *start* للبداية أو *end* للنهاية في المعادلة (القيمة الافتراضية هي *1* و *0* و *1* على التتالي). في نمط *RGBA* يمكن أن تكون قيمة *pname* تساوي *GL_FOG_COLOR* و في هذه الحالة يشير الوسيط *param* إلى قيم *RGBA* لتحديد لون الضباب.

تطبيقات عملية

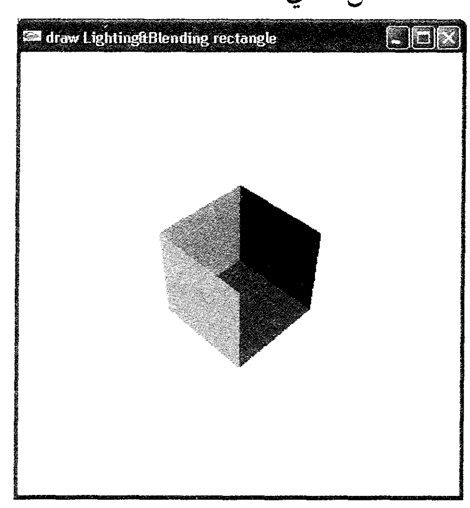
تطبيق 1

مزج المكعب المدروس في الفصل ٦ (التطبيق *application63*)، حيث يتم تفعيل المزج بضغط المفتاح *b* وإلغاء تفعيله بضغط المفتاح *d* . . أنشئ تطبيقاً اسمه *application71* وأنشئ ضمنه الملف *a18.cpp* وذلك بنسخ الملف *a16* من التطبيق *application63* إلى التطبيق *application64* ثم عدل اسم الملف ليصبح *a18.cpp*، ثم نفذ التعديلات التالية على الملف المنسوخ *a18.cpp* (التعديل فقط في التابع *keyboard*):

```
void keyboard ( unsigned char key, int x, int y )
{
  switch ( key ) {
    case 27:  /*  Escape key  */
      exit ( 0 );
      break;
    case 'l':
      glEnable(GL_LIGHTING);
      break;
    case 'f':
      glDisable(GL_LIGHTING);
      break;
   case 'b':
   glEnable(GL_BLEND);
   glDisable(GL_DEPTH_TEST);
      break;
   case 'd':
```

```
glDisable(GL_BLEND);
glEnable(GL_DEPTH_TEST);
      break;
  default:
    break;
      }
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



تطبيق 2

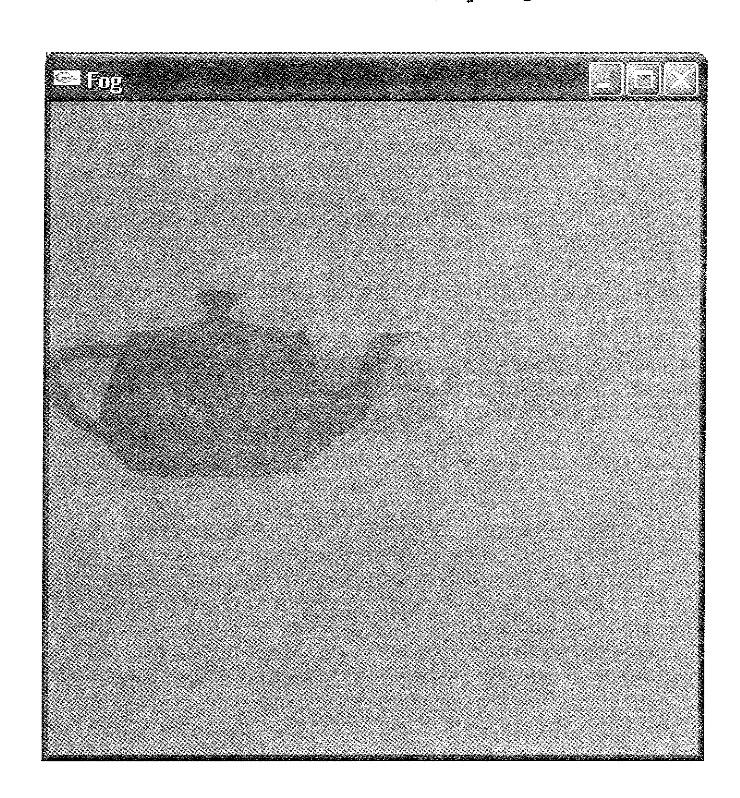
رسم خمسة أباريق شاي متتالية وتطبيق تأثير الضباب عليها. اضغط المفتاح *"l"* لتطبيق معادلة الضباب *GL_EXP* والمفتاح *"f"* لتطبيق معادلة الضباب *GL_EXP2* والمفتاح *"b"* لتطبيق معادلة الضباب *GL_LINEAR* . أنشئ تطبيقاً اسمه *application72* وأنشئ ضمنه الملف *a19.cpp* ، ثم اكتب النص البرمجي التالي ضمن الملف *a18.cpp* :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#include <math.h>
#include <gl\glaux.h>
GLint fogMode;
static void redraw(void);
void keyboard ( unsigned char key, int x, int y )
```

```
{
  switch ( key ) {
    case 27:  /*  Escape key  */
      exit ( 0 );
      break;
    case 'l':
      fogMode = GL_EXP2;
        glFogi (GL_FOG_MODE, fogMode);
      break;
    case 'f':
     fogMode = GL_EXP;
       glFogi (GL_FOG_MODE, fogMode);
      break;
   case 'b':
   fogMode = GL_LINEAR;
   glFogf (GL_FOG_START, 1.0);
     glFogf (GL_FOG_END, 15.0);
   glFogi (GL_FOG_MODE, fogMode);
          break;
        default:
       break;
         }
}
void myinit(void)
{
   glEnable(GL_DEPTH_TEST);
   glDepthFunc(GL_LEQUAL);
   glEnable(GL_FOG);
   {
      GLfloat density;
      GLfloat fogColor[4] = {0.5, 0.5, 0.5, 1.0};
      fogMode = GL_EXP;
      glFogi (GL_FOG_MODE, fogMode);
      glFogfv (GL_FOG_COLOR, fogColor);
      glFogf (GL_FOG_DENSITY, 0.1);
      glHint (GL_FOG_HINT, GL_DONT_CARE);
      glClearColor(0.5, 0.5, 0.5, 1.0);
   }
}
int main(int argc, char **argv)
{
```

```
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("Fog");
 myinit();
 glutDisplayFunc(redraw);
 glutKeyboardFunc    ( keyboard );
 glMatrixMode(GL_PROJECTION);
 gluPerspective(45,1.0,10.0,200.0);
 glMatrixMode(GL_MODELVIEW);
 glutMainLoop();
 return 0;
}
static void redraw(void)
{
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glColor3f(1.0, 0.0, 0.0);
 glTranslatef(-2.0f,0.5f,-10.0f);
 auxSolidTeapot (1.5);
 glTranslatef(2.0f,0.5f,-10.0f);
 auxSolidTeapot (1.5);
 glTranslatef(6.0f,0.5f,-10.0f);
 auxSolidTeapot (1.5);
 glTranslatef(10.0f,0.5f,-10.0f);
 auxSolidTeapot (1.5);
 glTranslatef(14.0f,0.5f,-10.0f);
 auxSolidTeapot (1.5);
 glutPostRedisplay();
 glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي (حالة تطبيق معادلة الضباب *GL_EXP2*):



الفصل الثامن

رسم النقاط الضوئية و الصور النقطية و الخطوط و الصور

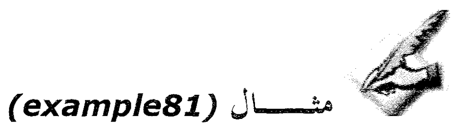
ستتعلم في هذا الفصل المواضيع التالية

- الصور النقطية و الخطوط ***Bitmaps and Fonts***
- الصور ***Images***
- تطبيقات عملية

الصور النقطية و الخطوط Bitmaps and Fonts

الصور النقطية عبارة عن مصفوفة مستطيلة من الأصفار و الواحدات تعمل كقناع رسم للجزء المستطيل الموافق لها من الإطار. لنفرض أنك رسمت صورة نقطية و كان اللون الحالي الأحمر. كل قيمة *1* في الصورة النقطية ستستبدل النقطة الضوئية الموافقة لها بنقطة ضوئية حمراء. إذا كانت القيمة *0* في الصورة النقطية، فلن تتأثر محتويات النقطة الضوئية. أكثر الاستخدامات الشائعة للصورة النقطية تتمثل في رسم الرموز على الشاشة.

تؤمن *OpenGL* أدنى مستوى من الدعم لرسم سلسلة من الرموز و معالجة الخطوط. يستخدم الأمران *glRasterPos*()* و *glBitmap()* لتوضيع و رسم صورة نقطية وحيدة على الشاشة.



مثـــال (example81)

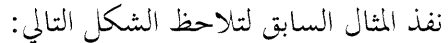
سنعمل ضمن هذا المثال على رسم الرمز *F* ثلاث مرات على الشاشة. يظهر الشكل التالي الرمز *F* كصورة نقطية مؤلفة من *12* سطر و *16* عمود بالإضافة إلى بيانات الصورة النقطية الموافقة لهذا الرمز.

```
0xff, 0xc0
0xff, 0xc0
0xc0, 0x00
0xc0, 0x00
0xc0, 0x00
0xff, 0x00
0xff, 0x00
0xc0, 0x00
0xc0, 0x00
0xc0, 0x00
0xc0, 0x00
0xc0, 0x00
```

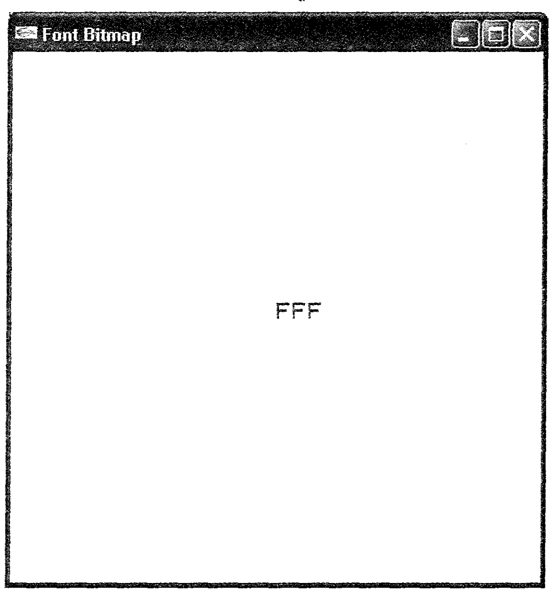
و لنكتب الآن المثال (الملف *e81.cpp*):

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
```

```
#include <gl\glut.h>
    تعبئ المصفوفة من الزاوية اليسارية السفلية//
GLubyte rasters[24] = {
0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00,
0xc0, 0x00, 0xff, 0x00, 0xff, 0x00, 0xc0, 0x00,
0xc0, 0x00, 0xc0, 0x00, 0xff, 0xc0, 0xff, 0xc0};
static void redraw(void);
void myinit(void)
{
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glClearColor(1.0, 1.0, 1.0, 0.0);
}
int main(int argc, char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
glutInitWindowPosition(100,100);
glutInitWindowSize(400,400);
glutCreateWindow("Font Bitmap");
myinit();
glutDisplayFunc(redraw);
glMatrixMode(GL_PROJECTION);
gluPerspective(45,1.0,10.0,200.0);
glMatrixMode(GL_MODELVIEW);
glutMainLoop();
return 0;
}
static void redraw(void)
{
glLoadIdentity();
glColor3f(1.0,0.0,0.0);
glTranslatef(0.0,0.0,-40);
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glRasterPos2i (0, 0);
glBitmap(10, 12, 0.0, 0.0, 12.0, 0.0, rasters);
glBitmap(10, 12, 0.0, 0.0, 12.0, 0.0, rasters);
glBitmap(10, 12, 0.0, 0.0, 12.0, 0.0, rasters);
glutSwapBuffers();
}
```



نفذ المثال السابق لتلاحظ الشكل التالي:



تلاحظ من الشكل السابق أن الجزء المرئي من الحرف *F* لا يتجاوز عرضه *10* نقطة ضوئية. تخزن بيانات الصور النقطية دائماً في قطع كبيرة تكون من مضاعفات *8* خانات. لكن عرض الصورة النقطية الحقيقي يمكن ألا يكون من مضاعفات *8*. يتم رسم الخانات الصانعة للصورة النقطية بدءً من الزاوية اليسارية السفلية، حيث يرسم أولاً السطر السفلي ثم السطر الأعلى منه و هكذا، و تلاحظ وجود هذا الترتيب في النص البرمجي السابق. تبدأ مصفوفة التخزين بـــ *0xc0, 0x00, 0xc0, 0x00* للسطرين السفليين للحرف ثم *0xff, 0xc0, 0xff, 0xc0* للسطرين التاليين وهكذا. أهم أمرين في هذا المثال هما *glRasterPos2i()* و *glBitmap()* وسنناقشهما بالتفصيل في الفقرة التالية.

موقع المسح الحالي

يمثل موقع المسح *Raster* الحالي المنطقة التي سترسم فيها الصورة النقطة التالية. في مثال الحرف *F* ، يعين موقع المسح باستخدام الأمر *glRasterPos*()* إلى الموقع *(20,20)* الذي ترسم فيه الزاوية اليسارية السفلية من الحرف *F*.

الأمر المذكور في المثال:

```
glRasterPos2i(20, 20);
```

الشكل العام لأمر تعيين موقع المسح:

```
void glRasterPos{234}{sifd}{v}(TYPE x, TYPE y, TYPE z, TYPE w);
```

تحدد المعاملات *x,y,z,w* إحداثيات موقع المسح. القيم الافتراضية لـــ *z* و *w* هي *z=0* و *w=1*.

تستطيع الحصول على الموقع الحالي للمسح باستخدام الأمر *glGetFloatv()* مع الثابت *GL_CURRENT_RASTER_POSITION* كوسيط أول ، أما الوسيط الثاني فيجب أن يكون مؤشراً إلى مصفوفة محددة تحوي قيم *x,y,z,w* كأرقام فاصلة عائمة.

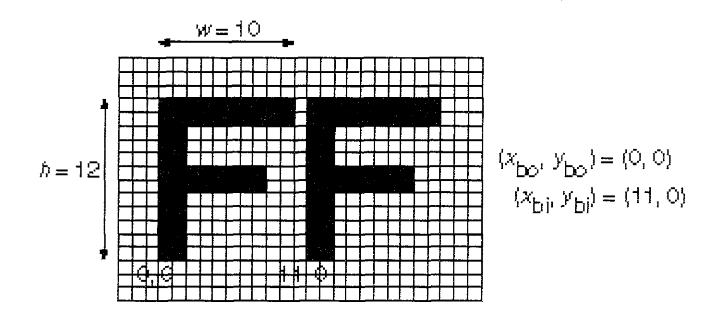
رسم الصورة النقطية

بعد أن تحدد موقع المسح المطلوب، يمكنك استخدام الأمر *glBitmap()* لرسم البيانات.

الشكل العام لأمر رسم الصورة النقطية :

```
void glBitmap(GLsizei width, GLsizei height, GLfloat xbo, GLfloat
ybo, GLfloat xbi, GLfloat ybi, const GLubyte *bitmap);
```

يرسم الصورة المحددة بالمؤشر *bitmap* الذي يشير إلى الصورة النقطية. توضع الصورة النقطية بدءً من موقع المسح المعرف حديثاً. تشير معاملات *width* و *height* إلى عرض و ارتفاع الصورة النقطية بالنقطة الضوئية. ليس من الضروري أن يكون العرض من مضاعفات *8*، على الرغم من أن البيانات تخزن على شكل رموز غير مؤشرة بحجم *8* خانة. استخدم *xbo* و *ybo* لتحديد موقع الصورة النقطية (القيم الموجبة تزيح الموقع إلى الأعلى و اليمين، أما القيم السالبة فتزيح موقع الصورة إلى الأسفل و اليسار). تشير *xbi* و *ybi* إلى الزيادة في *x* و *y* التي يجب إضافتها إلى موقع المسح بعد مسح الصورة كما هو مبين في الشكل التالي



بعد الانتهاء من الرسم، يزداد موقع المسح بقيمة *xbi* و *ybi* بالاتجاهين *x* و *y* بشكل متتالي. من أجل الخطوط الأجنبية، تكون قيمة *ybi* مساوية لـــ *0.0* و قيمة *xbi* موجبة. أما في حالة الخطوط العربية حيث تكتب الحروف من اليمين إلى اليسار، فتكون قيم *xbi* سالبة. من أجل الخطوط التي ترسم بشكل شاقولي ضمن أعمدة، فستكون *xbi=0* و قيم *ybi* غير صفرية. في الشكل السابق رسم الحرف *F* الأول ثم أزيح موقع المسح بمقدار *12* نقطة ضوئية، للسماح يترك فراغ مقداره *2* نقطة ضوئية بين الحروف ثم رسم الحرف *F* الثاني. يحفظ موقع المسح الحالي بصيغة فاصلة عائمة، و بالتالي يرسم الرمز بالقرب من المنطقة المحددة في موقع المسح قدر الإمكان. في المثال السابق الخاص بالحرف *F*، لو عدلنا قيمة *xbi* بحيث نجعلها *11.5* بدلاً من *12* و أردنا بعد ذلك رسم رموز أخرى، فستتناوب عند ذلك الفراغات بين الرموز ما بين نقطة و نقطتين ضوئيتين. لاحظ أنه لا يمكن استخدام الصور النقطية لرسم خطوط قابلة للتدوير لأن الصور النقطية ترسم دائماً بمحاذاة محاور الذاكرة المؤقتة للإطار.

الخطوط و لوائح الإظهار

يتألف الخط من مجموعة رموز، حيث يملك كل رمز رقم تعريف خاص به (عادة رمز آسكي) و طريقة رسم خاصة به. من أجل مجموعة آسكي القياسية، يملك الحرف *A* شيفرة آسكي *65* و *B* القيمة *66* و هكذا.تمثل السلسلة النصية *"DAB"* بالقيمة *68,65,66*. يمكن تمثيل ذلك بلوائح إظهار، حيث ترسم اللائحة ذات الرقم *65* الحرف *A* و اللائحة ذات

الرقم *66* الحرف *B* و هكذا. عندما نرسم السلسلة النصية *68,65,66* فما علينا سوى تنفيذ لوائح الإظهار الموافقة لتلك الأرقام.

يمكنك استخدام أمر *glCallLists()* الذي له الشكل العام التالي:

```
void glCallLists(GLsizei n, GLenum type, const GLvoid *lists);
```

يشير الوسيط الأول *n* إلى رقم الرمز المراد رسمه، أما*type* فقيمتها عادة *GL_BYTE*، أما *lists* فهي عبارة عن مصفوفة تحوي تشفير الرموز.

تحتاج الكثير من التطبيقات إلى رسم سلاسل الرموز بخطوط متعددة و أحجام مختلفة. تستطيع بدلاً من استخدام الرقم *65* لتمثيل الحرف *A*، إجبار الخط الأول *Font1* على تشفير الحروف *...A,B,C* بالأرقام *...1065,1066,1067* و الخط الثاني *Font2* بالأرقام *...2065,2066,2067* ، لكن أي أرقام أكبر من *256* لن تتسع ضمن بايت واحد. الحل الأنسب يتمثل بإضافة إزاحة إلى كل مدخل (رمز) في السلسلة ثم اختيار لائحة الإظهار المناسبة. في حالتنا تمثل الأحرف *A,B,C* ضمن الخط *Font1* بالأرقام *1065,1066,1067*على التتالي و قد تمثل في الخط *Font2* بـــ *2065,2066,2067*. بعد ذلك، لرسم رمز بالخط *Font1*، يتم تعيين القيمة *1000* إلى الإزاحة و رسم لوائح الإظهار *65,66,67* . لرسم نفس السلسلة بالخط *Font2*، نعين القيمة *2000* إلى الإزاحة و ترسم نفس لوائح الإظهار. استخدم الأمر *glListBase()* لتعيين الإزاحة. يجب أن يستدعى هذا الأمر من أجل المثال السابق بالقيمة *1000* أو *2000* كوسائط. ما تحتاج الآن لمعرفته هو عدد لوائح الإظهار غير المستخدمة، التي يمكنك الحصول عليها من خلال الأمر *glGenLists()* الذي له الشكل العام التالي:

```
GLuint glGenLists(GLsizei range);
```

يعيد الأمر السابق مجال لوائح الإظهار المعرفة. اللوائح المعادة تعلم كلوائح مستخدمة حتى إذا كانت فارغة، لذلك فإن الاستدعاء المتتالي للأمر *glGenLists()* لن يعيد مطلقاً نفس اللوائح (ما لم تكن قد حذفتها مسبقاً). لنفرض أنك كنت تستخدم الرقم *4* كوسيط و لنفرض أن التابع *glGenLists()* قد أعاد القيمة *81*، فبإمكانك استخدام لوائح الإظهار

81,82,83,84 من أجل رموزك. إذا لم يتمكن الأمر *glGenLists()* من إيجاد معرفات لوائح غير مستخدمة للطول المطلوب، فسيعيد القيمة *0* (يستطيع الأمر *glDeleteLists()* حذف جميع اللوائح المرافقة لخط ما بعملية واحدة).

تمتلك معظم الخطوط الأمريكية و الأوربية عدد رموز قليل (أقل من *256*) مما يتيح إمكانية تمثيل كل رمز بشيفرة مختلفة ضمن بايت واحد. قد تحتاج الخطوط في آسيا إلى مجموعات رمز أكبر، لذلك تكون عملية تشفير رمز في البايت الواحد مستحيلة. يسمح للسلاسل بأن تتألف من رموز بطول *1* أو *2* أ، *3* أو *4* بايت من خلال الوسيط *type* ضمن الأمر *glCallLists()* . يمكن أن يمتلك هذا الوسيط إحدى القيم التالية:

GL_BYTE GL_UNSIGNED_BYTE
GL_SHORT GL_UNSIGNED_SHORT
GL_INT GL_UNSIGNED_INT
GL_FLOAT GL_2_BYTES
GL_3_BYTES GL_4_BYTES

الصور ***Images***

تشبه الصورة *Image* الصورة النقطية *Bitmap*، و لكن بدلاً من احتواء الصورة النقطية على خانة وحيدة من أجل كل نقطة ضوئية في منطقة مستطيلة من الشاشة، يمكن أن تحوي الصورة *Image* معلومات أكثر. على سبيل المثال يمكن أن تحوي صورة القيمة *(R,G,B,A)* الكاملة لكل نقطة ضوئية. يمكن أن تأتي الصور من مصادر متعددة مثل مسح صورة فوتوغرافية بواسطة الماسح الضوئي، أو من خلال توليد صورة على الشاشة بواسطة برنامج رسوميات باستخدام التجهيزات الرسومية، ثم إعادة قراءة الصورة نقطة ضوئية تلو الأخرى، أو من خلال برنامج يولد الصورة في الذاكرة نقطة ضوئية تلو الأخرى. يمكن استخدام الصور من أجل خرائط الإكساء.

كتابة و قراءة ونسخ بيانات النقاط الضوئية

تؤمن *OpenGL* ثلاثة أوامر تتعامل مع بيانات الصور:

- ***glReadPixels()*** : يقرأ مصفوفة مستطيلة من النقاط الضوئية من الذاكرة المؤقتة للإطار و يحفظ بياناتها ضمن ذاكرة الحاسب.
- ***glDrawPixels()*** : يكتب مصفوفة مستطيلة من النقاط الضوئية ضمن الذاكرة المؤقتة للإطار و ذلك من البيانات المحفوظة في ذاكرة الحاسب.
- ***glCopyPixels()*** : ينسخ مصفوفة مستطيلة من النقاط الضوئية من مكان في الذاكرة المؤقتة للإطار إلى مكان آخر من نفس الذاكرة المؤقتة.

يمكن أن يتصرف هذا الأمر بشكل مشابه لاستدعاء *glReadPixels()*، ثم استدعاء *glDrawPixels()*، لكن باستخدام *glCopyPixels()* لا تكتب البيانات مطلقاً ضمن ذاكرة الحاسب. و لنبدأ بشرح الأوامر السابقة بشيء من التفصيل:

الأمر *glReadPixels()*

الشكل العام لهذا الأمر:

*void glReadPixels(GLint x, GLint y, GLsizei width, GLsizei height, GLenum format, GLenum type, GLvoid *pixels);*

يقرأ بيانات النقاط الضوئية من منطقة مستطيلة في الذاكرة المؤقتة للإطار. إحداثيات الزاوية اليسارية السفلية لهذه المنطقة متمثلة بقيم *(x,y)* و أبعادها متمثلة بقيم *width* (العرض) و*height* (الارتفاع)، و مخزنة ضمن المصفوفة المشار إليها بواسطة *pixels* . يشير الوسيط *format* إلى نوع عناصر بيانات النقاط الضوئية التي يتم قراءتها (يمكن أن تكون قيم فهرس لون أو قيمة *R,G,B,A*)، أما الوسيط *type* فيشير إلى نوع البيانات لكل عنصر. يعرض الجدول التالي القيم المتحملة للوسيط *format* (نوع عناصر بيانات النقاط الضوئية):

الاسم	نوع عناصر بيانات النقاط الضوئية
GL_COLOR_INDEX	قيمة فهرس لون وحيدة .
GL_RGB	مكون اللون الأحمر يليه الأخضر يليه الأزرق
GL_RGBA	مكون اللون الأحمر يليه الأخضر يليه الأزرق يليه قيمة الشفافية ألفا
GL_RED	مكون لون أحمر وحيد
GL_GREEN	مكون لون أخضر وحيد
GL_BLUE	مكون لون أزرق وحيد
GL_ALPHA	مكون لون شفافية وحيد
GL_LUMINANCE	مكون إضاءة ذاتية وحيد
GL_LUMINANCE_ALPHA	مكون إضاءة ذاتية يليه مكون شفافية
GL_STENCIL_INDEX	فهرس حاجب وحيد
GL_DEPTH_COMPONENT	مكون عمق وحيد

يبين الجدول التالي القيم المحتملة للوسيط *type* (نوع بيانات كل عنصر):

الاسم	نوع البيانات
GL_UNSIGNED_BYTE	عدد صحيح غير مؤشر بطول *8 bit*
GL_BYTE	عدد صحيح مؤشر بطول *8 bit*
GL_BITMAP	خانات وحيدة ضمن أعداد صحيحة غير مؤشرة بطول *8 bit*
GL_UNSIGNED_SHORT	عدد صحيح غير مؤشر بطول *16 bit*
GL_SHORT	عدد صحيح مؤشر بطول *16 bit*
GL_UNSIGNED_INT	عدد صحيح غير مؤشر بطول *32 bit*
GL_INT	عدد صحيح بطول *32 bit*
GL_FLOAT	عدد فاصلة عائمة أحادي الدقة

الأمر *glDrawPixels()*

الشكل العام لهذا الأمر:

*void glDrawPixels(GLsizei width, GLsizei height, GLenum format, GLenum type, const GLvoid *pixels);*

يرسم هذا الأمر مستطيلاً من بيانات النقاط الضوئية عرضه *width* و ارتفاعه *height* . يرسم المستطيل بتحديد الزاوية السفلية اليسارية له التي تتوضع عند موقع المسح الحالي. يملك الوسيطان *format* و *type* نفس المعنى المذكور في الأمر *glReadPixels()*

و نفس القيم المذكورة في الجدولين السابقين. تحوي المصفوفة المشار إليها بالوسيط *pixels* بيانات النقاط الضوئية المراد رسمها. إذا كانت قيمة موقع المسح الحالي غير صحيحة فلن يرسم شيء.

تذكر أنه بالاعتماد على الوسيط *format*، يمكن قراءة أو كتابة من عنصر واحد و حتى أربعة عناصر. على سبيل المثال، إذا كانت قيمة الوسيط *format* مساوية إلى *GL_RGBA* و كنت تقرأ إلى أعداد صحيحة بطول *32 bit*(أي أن قيمة *type=* *GL_INT*)، و نتيجة لذلك فإن كل نقطة ضوئية تُقرأ ستحتاج إلى *16 bytes* لتخزينها (لدينا *4* مكونات × *4* بايت لكل مكون).

الأمر *glCopyPixels()*

الشكل العام لهذا الأمر:

void glCopyPixels(GLint x, GLint y, GLsizei width, GLsizei height, GLenum type);

ينسخ بيانات نقاط ضوئية من منطقة مستطيلة في الذاكرة المؤقتة للإطار، إحداثيات الزاوية اليسارية السفلية لهذه الزاوية متمثلة بقيم *(x,y)* وعرضها *width* و ارتفاعها *height* . تنسخ البيانات إلى موقع جديد إحداثيات الزاوية اليسارية السفلية له معطاة بموقع المسح الحالي. قيمة *type* إما *GL_COLOR* أو *GL_STENCIL* أو *GL_DEPTH* . يتصرف الأمر *glCopyPixels()* بشكل مشابه لاستخدام الأمر *glReadPixels()* يليه الأمر *glDrawPixels()* مع التفسير التالي من أجل الوسيطان *format* و *type*.

- إذا كانت قيمة *type* للأمر *glCopyPixels()* هي *GL_DEPTH* أو *GL_STENCIL* ، يمكن استخدام القيم *GL_DEPTH_COMPONENT* أو *GL_STENCIL_INDEX* من أجل الوسيط *format* للأمرين *glReadPixels()* و *glDrawPixels()* .

- إذا كانت قيمة *type* هي *GL_COLOR* ، يمكن عند ذلك استخدام *GL_RGBA* أو *GL_COLOR_INDEX* حسب حالة نمط النظام هل هي *R,G,B,A* أم فهرس اللون.

لاحظ أنه لا حاجة لاستخدام الوسيط *format* لأن البيانات لا تنسخ مطلقاً إلى ذاكرة الحاسب.

توسيع أو إنقاص حجم الصورة

بشكل اعتيادي كل نقطة ضوئية في الصورة تكتب إلى نقطة ضوئية و حيدة على الشاشة. من ناحية أخرى يمكنك و بشكل عشوائي زيادة أو إنقاص حجم صورة باستخدام الأمر *glPixelZoom()* الذي له الشكل العام التالي:

```
void glPixelZoom(GLfloat zoomx, GLfloat zoomy);
```

يستخدم الوسيطان *zoomx* و *zoomy* لتكبير أو تصغير عمليات كتابة النقاط الضوئية على الشاشة و فق أبعاد *x,y*.

قيمة *zoomx* و *zoomy* الافتراضية هي *1.0* . إذا وضعت القيمة *2.0* لكليهما، فستكتب كل نقطة ضوئية من الصورة إلى *4* نقاط ضوئية على الشاشة.

يمكن استخدام قيمة كسرية لهذين الوسيطين و كذلك قيم سالبة.

تطبيقات عملية

تطبيق *1*

تعريف واستخدام خط كامل

يعمل هذا التطبيق على تعريف خط كامل باستخدام الأمر *glBitmap()* وتقنية لوائح الإظهار .

سنعرف خط آسكي كامل يملك فيه كل رمز نفس العرض . يشبه هذا التطبيق المثال السابق (مثال الحرف *F*) إلا أنه في هذا التطبيق تم استخدام *95* صورة نقطية *Bitmap* مختلفة (صورة نقطية لكل رمز آسكي بما في ذلك رمز الفراغ). كل معرف لائحة إظهار

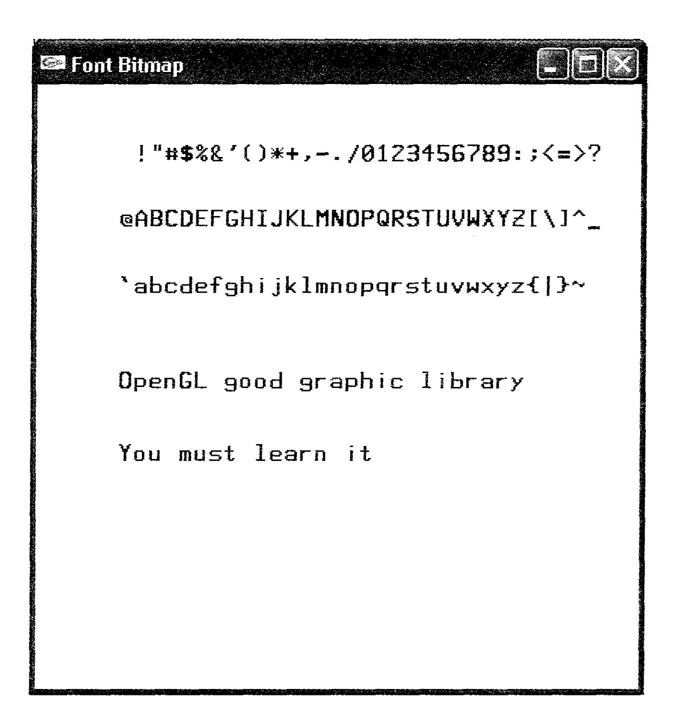
لرمز معين يكافئ شيفرة الآسكي لذلك الرمز . أنشئ تطبيقاً اسمه *application81* وأنشئ ضمنه الملف *a20.cpp* ،ثم اكتب النص البرمجي التالي ضمن الملف *a20.cpp* :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
GLubyte rasters[][13] = {
{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0x18, 0x18, 0x00, 0x00, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18},
{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x36, 0x36, 0x36, 0x36},
{0x00, 0x00, 0x00, 0x66, 0x66, 0xff, 0x66, 0x66, 0xff, 0x66, 0x66, 0x00, 0x00},
{0x00, 0x00, 0x18, 0x7e, 0xff, 0x1b, 0x1f, 0x7e, 0xf8, 0xd8, 0xff, 0x7e, 0x18},
{0x00, 0x00, 0x0e, 0x1b, 0xdb, 0x6e, 0x30, 0x18, 0x0c, 0x76, 0xdb, 0xd8, 0x70},
{0x00, 0x00, 0x7f, 0xc6, 0xcf, 0xd8, 0x70, 0x70, 0xd8, 0xcc, 0xcc, 0x6c, 0x38},
{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x1c, 0x0c, 0x0e},
{0x00, 0x00, 0x0c, 0x18, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x18, 0x0c},
{0x00, 0x00, 0x30, 0x18, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x18, 0x30},
{0x00, 0x00, 0x00, 0x00, 0x99, 0x5a, 0x3c, 0xff, 0x3c, 0x5a, 0x99, 0x00, 0x00},
{0x00, 0x00, 0x00, 0x18, 0x18, 0x18, 0xff, 0xff, 0x18, 0x18, 0x18, 0x00, 0x00},
{0x00, 0x00, 0x30, 0x18, 0x1c, 0x1c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0x00, 0x38, 0x38, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x60, 0x60, 0x30, 0x30, 0x18, 0x18, 0x0c, 0x0c, 0x06, 0x06, 0x03, 0x03},
{0x00, 0x00, 0x3c, 0x66, 0xc3, 0xe3, 0xf3, 0xdb, 0xcf, 0xc7, 0xc3, 0x66, 0x3c},
{0x00, 0x00, 0x7e, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x78, 0x38, 0x18},
{0x00, 0x00, 0xff, 0xc0, 0xc0, 0x60, 0x30, 0x18, 0x0c, 0x06, 0x03, 0xe7, 0x7e},
{0x00, 0x00, 0x7e, 0xe7, 0x03, 0x03, 0x07, 0x7e, 0x07, 0x03, 0x03, 0xe7, 0x7e},
{0x00, 0x00, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0xff, 0xcc, 0x6c, 0x3c, 0x1c, 0x0c},
{0x00, 0x00, 0x7e, 0xe7, 0x03, 0x03, 0x07, 0xfe, 0xc0, 0xc0, 0xc0, 0xc0, 0xff},
{0x00, 0x00, 0x7e, 0xe7, 0xc3, 0xc3, 0xc7, 0xfe, 0xc0, 0xc0, 0xc0, 0xe7, 0x7e},
{0x00, 0x00, 0x30, 0x30, 0x30, 0x30, 0x18, 0x0c, 0x06, 0x03, 0x03, 0x03, 0xff},
{0x00, 0x00, 0x7e, 0xe7, 0xc3, 0xc3, 0xe7, 0x7e, 0xe7, 0xc3, 0xc3, 0xe7, 0x7e},
{0x00, 0x00, 0x7e, 0xe7, 0x03, 0x03, 0x03, 0x7f, 0xe7, 0xc3, 0xc3, 0xe7, 0x7e},
{0x00, 0x00, 0x00, 0x38, 0x38, 0x00, 0x00, 0x38, 0x38, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0x30, 0x18, 0x1c, 0x1c, 0x00, 0x00, 0x1c, 0x1c, 0x00, 0x00, 0x00},
{0x00, 0x00, 0x06, 0x0c, 0x18, 0x30, 0x60, 0xc0, 0x60, 0x30, 0x18, 0x0c, 0x06},
{0x00, 0x00, 0x00, 0x00, 0xff, 0xff, 0x00, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0x60, 0x30, 0x18, 0x0c, 0x06, 0x03, 0x06, 0x0c, 0x18, 0x30, 0x60},
{0x00, 0x00, 0x18, 0x00, 0x00, 0x18, 0x18, 0x0c, 0x06, 0x03, 0xc3, 0xc3, 0x7e},
{0x00, 0x00, 0x3f, 0x60, 0xcf, 0xdb, 0xd3, 0xdd, 0xc3, 0x7e, 0x00, 0x00, 0x00},
{0x00, 0x00, 0xc3, 0xc3, 0xc3, 0xc3, 0xff, 0xc3, 0xc3, 0xc3, 0x66, 0x3c, 0x18},
{0x00, 0x00, 0xfe, 0xc7, 0xc3, 0xc3, 0xc7, 0xfe, 0xc7, 0xc3, 0xc3, 0xc7, 0xfe},
{0x00, 0x00, 0x7e, 0xe7, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xe7, 0x7e},
{0x00, 0x00, 0xfc, 0xce, 0xc7, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc7, 0xce, 0xfc},
{0x00, 0x00, 0xff, 0xc0, 0xc0, 0xc0, 0xc0, 0xfc, 0xc0, 0xc0, 0xc0, 0xc0, 0xff},
{0x00, 0x00, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xfc, 0xc0, 0xc0, 0xc0, 0xff},
{0x00, 0x00, 0x7e, 0xe7, 0xc3, 0xc3, 0xcf, 0xc0, 0xc0, 0xc0, 0xc0, 0xe7, 0x7e},
{0x00, 0x00, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xff, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3},
{0x00, 0x00, 0x7e, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x7e},
{0x00, 0x00, 0x7c, 0xee, 0xc6, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06},
{0x00, 0x00, 0xc3, 0xc6, 0xcc, 0xd8, 0xf0, 0xe0, 0xf0, 0xd8, 0xcc, 0xc6, 0xc3},
{0x00, 0x00, 0xff, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0},
{0x00, 0x00, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xdb, 0xff, 0xff, 0xe7, 0xc3},
{0x00, 0x00, 0xc7, 0xc7, 0xcf, 0xcf, 0xdf, 0xdb, 0xfb, 0xf3, 0xf3, 0xe3, 0xe3},
```

```
{0x00, 0x00, 0x7e, 0xe7, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xe7, 0x7e},
{0x00, 0x00, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xfe, 0xc7, 0xc3, 0xc3, 0xc7, 0xfe},
{0x00, 0x00, 0x3f, 0x6e, 0xdf, 0xdb, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0x66, 0x3c},
{0x00, 0x00, 0xc3, 0xc6, 0xcc, 0xd8, 0xf0, 0xfe, 0xc7, 0xc3, 0xc3, 0xc7, 0xfe},
{0x00, 0x00, 0x7e, 0xe7, 0x03, 0x03, 0x07, 0x7e, 0xe0, 0xc0, 0xc0, 0xe7, 0x7e},
{0x00, 0x00, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0xff},
{0x00, 0x00, 0x7e, 0xe7, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3},
{0x00, 0x00, 0x18, 0x3c, 0x3c, 0x66, 0x66, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3},
{0x00, 0x00, 0xc3, 0xe7, 0xff, 0xff, 0xdb, 0xdb, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3},
{0x00, 0x00, 0xc3, 0x66, 0x66, 0x3c, 0x3c, 0x18, 0x3c, 0x3c, 0x66, 0x66, 0xc3},
{0x00, 0x00, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3c, 0x3c, 0x66, 0x66, 0xc3},
{0x00, 0x00, 0xff, 0xc0, 0xc0, 0x60, 0x30, 0x7e, 0x0c, 0x06, 0x03, 0x03, 0xff},
{0x00, 0x00, 0x3c, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x3c},
{0x00, 0x03, 0x03, 0x06, 0x06, 0x0c, 0x0c, 0x18, 0x18, 0x30, 0x30, 0x60, 0x60},
{0x00, 0x00, 0x3c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x3c},
{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc3, 0x66, 0x3c, 0x18},
{0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x38, 0x30, 0x70},
{0x00, 0x00, 0x7f, 0xc3, 0xc3, 0x7f, 0x03, 0xc3, 0x7e, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0xfe, 0xc3, 0xc3, 0xc3, 0xc3, 0xfe, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0},
{0x00, 0x00, 0x7e, 0xc3, 0xc0, 0xc0, 0xc0, 0xc3, 0x7e, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0x7f, 0xc3, 0xc3, 0xc3, 0xc3, 0x7f, 0x03, 0x03, 0x03, 0x03, 0x03},
{0x00, 0x00, 0x7f, 0xc0, 0xc0, 0xfe, 0xc3, 0xc3, 0x7e, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0x30, 0x30, 0x30, 0x30, 0x30, 0xfc, 0x30, 0x30, 0x30, 0x33, 0x1e},
{0x7e, 0xc3, 0x03, 0x03, 0x7f, 0xc3, 0xc3, 0xc3, 0x7e, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xfe, 0xc0, 0xc0, 0xc0, 0xc0},
{0x00, 0x00, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x00, 0x00, 0x18, 0x00},
{0x38, 0x6c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x00, 0x00, 0x0c, 0x00},
{0x00, 0x00, 0xc6, 0xcc, 0xf8, 0xf0, 0xd8, 0xcc, 0xc6, 0xc0, 0xc0, 0xc0, 0xc0},
{0x00, 0x00, 0x7e, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x78},
{0x00, 0x00, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xfe, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xfc, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0x7c, 0x00, 0x00, 0x00, 0x00},
{0xc0, 0xc0, 0xc0, 0xfe, 0xc3, 0xc3, 0xc3, 0xc3, 0xfe, 0x00, 0x00, 0x00, 0x00},
{0x03, 0x03, 0x03, 0x7f, 0xc3, 0xc3, 0xc3, 0xc3, 0x7f, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xe0, 0xfe, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0xfe, 0x03, 0x03, 0x7e, 0xc0, 0xc0, 0x7f, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0x1c, 0x36, 0x30, 0x30, 0x30, 0x30, 0xfc, 0x30, 0x30, 0x30, 0x00},
{0x00, 0x00, 0x7e, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0x18, 0x3c, 0x3c, 0x66, 0x66, 0xc3, 0xc3, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0xc3, 0xe7, 0xff, 0xdb, 0xc3, 0xc3, 0xc3, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0xc3, 0x66, 0x3c, 0x18, 0x3c, 0x66, 0xc3, 0x00, 0x00, 0x00, 0x00},
{0xc0, 0x60, 0x60, 0x30, 0x18, 0x3c, 0x66, 0x66, 0xc3, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0xff, 0x60, 0x30, 0x18, 0x0c, 0x06, 0xff, 0x00, 0x00, 0x00, 0x00},
{0x00, 0x00, 0x0f, 0x18, 0x18, 0x18, 0x38, 0xf0, 0x38, 0x18, 0x18, 0x18, 0x0f},
{0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18},
{0x00, 0x00, 0xf0, 0x18, 0x18, 0x18, 0x1c, 0x0f, 0x1c, 0x18, 0x18, 0x18, 0xf0},
{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x8f, 0xf1, 0x60, 0x00, 0x00, 0x00}
};
GLuint fontOffset;
static void redraw(void);
int main(int argc, char **argv);
void makeRasterFont(void)
{
 GLuint i;
 glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
 fontOffset = glGenLists (128);
```

```
  for (i = 32; i < 127; i++) {
   glNewList(i+fontOffset, GL_COMPILE);
   glBitmap(8, 13, 0.0, 2.0, 10.0, 0.0, rasters[i-32]);
   glEndList();
  }
}
void myinit(void)
{
 glShadeModel (GL_FLAT);
 makeRasterFont();
}
void printString(char *s)
{
 glPushAttrib (GL_LIST_BIT);
 glListBase(fontOffset);
 glCallLists(strlen(s), GL_UNSIGNED_BYTE, (GLubyte *) s);
 glPopAttrib ();
}
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("Font Bitmap");
 myinit();
 glutDisplayFunc(redraw);
glMatrixMode(GL_PROJECTION);
gluPerspective(45,1.0,10.0,200.0);
glMatrixMode(GL_MODELVIEW);
glutMainLoop();
return 0;
}
static void redraw(void)
{
 GLfloat blue[3] = { 0.0, 0.0, 1.0 };
 int i, j=0;
 char teststring[33];
 glClearColor(1.0,1.0,1.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glTranslatef(-20.0f,-150.0f,-200.0f);
 glColor3fv(blue);
 for (i = 32; i < 127; i += 32) {
   glRasterPos2i(-40, 230 - 18*i/32);
   for (j = 0; j < 32; j++)
      teststring[j] = (char) (i+j);
     teststring[32] = 0;
     printString(teststring);
}
 glRasterPos2i(-40,150);
 printString("OpenGL good graphic library ");
 glRasterPos2i(-40, 130);
 printString("You must learn it");
 glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



تطبيق 2

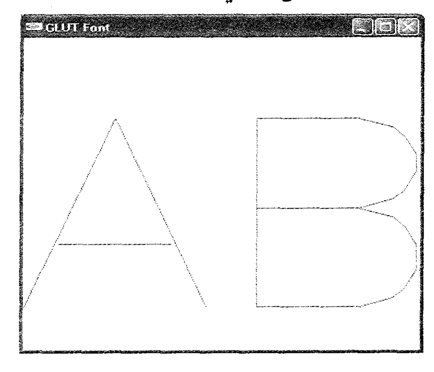
استخدام أوامر الخطوط المرافقة للمكتبة GLUT

يعمل هذا التطبيق على استخدام خطوط *Stroke* التي يتم فيها تصيير كل رمز على شكل مجموعة من الخطوط المقسمة *lines* (لمزيد من المعلومات راجع الملحق "ج"). أنشئ تطبيقاً اسمه *application82* وأنشئ ضمنه الملف *a21.cpp*، ثم اكتب النص البرمجي التالي ضمن الملف *a21.cpp* :

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#include <math.h>
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
```

```
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
 glutCreateWindow("GLUT Font");
 glutDisplayFunc(redraw);
 glMatrixMode(GL_PROJECTION);
 gluPerspective(45,1.0,10.0,300.0);
 glMatrixMode(GL_MODELVIEW);
 glutMainLoop();
 return 0;
}
static void redraw(void)
{
 glClearColor(1.0,1.0,1.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glTranslatef(-85.0f,-60.0f,-200.0f);
 glColor3f(1.0, 0.0, 0.0);
 //تعليمات رسم الرموز//
 glutStrokeCharacter(GLUT_STROKE_ROMAN,65);
 glutStrokeCharacter(GLUT_STROKE_MONO_ROMAN,66);
 glutSwapBuffers();
}
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



الفصل التاسع

إكساء العناصر الهندسية بالتراكيب
TEXTURES

ستتعلم في هذا الفصل المواضيع التالية

- مقدمة
- خطوات إكساء عنصر بتركيب ***Texture***
- تحديد تراكيب الإكساء
- إسناد إحداثيات التراكيب
- تكرار وحصر التراكيب ***Textures***
- تطبيقات عملية

مقدمة

تعلمت حتى الآن رسم العناصر الهندسية ثم تلوينها بلون واحد أو بألوان متعددة. لكن هذا لا يحاكي الواقع. و لكي تتمكن من جعل عناصرك الرسومية واقعية، يجب عليك إكساءها بالصورة (التركيب) المناسبة. على سبيل المثال إذا رسمنا كرة دون إكساء فستبدو كرة عادية لا تحاكي الواقع بشيء، أما إذا كسيناها بخريطة الكرة الأرضية فسنحصل على محاكاة للكرة الأرضية.

يمكن تطبيق الإكساء على سطح ما بعدة طرق. تتمثل إحدى الطرق بدهان التركيب على السطح مباشرة، هناك طريقة أخرى تتمثل باستخدام التركيب لتعديل لون الدهان للسطح، و هناك طريقة ثالثة تتمثل باستخدام التركيب لمزج لونه مع لون السطح.

خطوات إكساء عنصر بتركيب Texture

تعلمك هذه الفقرة الخطوات الضرورية لإنجاز الإكساء. و إليك هذه الخطوات:

١. تحديد التركيب *Texture*

أبسط حالة للتركيب عبارة عن صورة وحيدة. يتألف التركيب عادة من بعدين (مثل معظم الصور) كما يمكن أن يكون ببعد واحد. يمكن أن تتألف البيانات التي تصف التركيب من مكون واحد أو مكونين أو ثلاثة مكونات أو أربعة مكونات و ذلك حسب المركب اللوني *(R,G,B,A)*. هناك تقنية متقدمة تسمى *Mipmapping* تمكنك من تحديد تركيب وحيد بدقات مختلفة ، و هذا يسمح لك بزيادة الحجم الفعال للتراكيب المتاحة كبيرة الحجم.

٢. توجيه كيفية تطبيق التركيب على كل نقطة ضوئية

يمكنك الاختيار من بين ثلاثة توابع مستخدمة لحساب القيمة النهائية لـــ *RGBA* من لون *Fragment* و بيانات صورة التركيب. يتمثل التابع الأول باستخدام لون التركيب كلون نهائي، يدعى هذا بالنمط *decal* (النسخ) و فيه يدهن التركيب فوق *Fragment*. يتمثل التابع الثاني باستخدام التركيب لتعديل أو تغيير حجم لون *Fragment*، هذه التقنية

مفيدة لدمج تأثير الإضاءة مع الإكساء. يتمثل التابع الأخير بمزج لون ثابت بلون *Fragment* اعتماداً على قيمة التركيب.

٣. تفعيل الإكساء بالتركيب

أنت بحاجة لتفعيل الإكساء قبل رسم عناصر مشهدك. يمكن تفعيل الإكساء باستخدام الأمر *glEnable( )* مع الثابت *GL_TEXTURE_1D* من أجل التراكيب أحادية البعد، و الثابت *GL_TEXTURE_2D* من أجل التراكيب ثنائية البعد، كما يمكن إلغاء تفعيل الإكساء باستخدام الأمر *glDisable( )* مع الثوابت السابقة.

٤. رسم المشهد بتزويد إحداثيات التراكيب و الإحداثيات الهندسية

أنت بحاجة لتحديد إحداثيات التراكيب و الإحداثيات الهندسية، مثلما تحدد عناصرك في المشهد. يستفاد من هذه الإحداثيات في تحديد محاذاة التركيب بالنسبة للعنصر المطبق الإكساء عليه. على سبيل المثال، من أجل التراكيب ثنائية البعد، يكون مجال إحداثيات التراكيب من *0.0* إلى *1.0* في كلا الاتجاهين،لكن إحداثيات العناصر المراد إكساؤها يمكن أن تكون أي شيء. لنفرض مثلاً أنه لدينا جدار شكله مربع و أردنا إكساءه بنسخة تركيب واحدة (مثلاً تركيب يمثل قرميدة واحدة)، فستكون إحداثيات التركيب *(0,0)* للزاوية اليسارية السفلية و *(1,0)* للزاوية اليمنية السفلية و *(1,1)* للزاوية اليمنية العلوية و *(0,1)* للزاوية اليسارية العلوية. إذا كان الجدار كبيراً، فقد تحتاج إلى إكسائه بعدة نسخ من التركيب، يمكن عند ذلك وضع رقم التكرار وفق المحور المراد التكرار وفقه، ففي مثال الجدار نضع الرقم *2* بدلاً من *1* لتكرار نسختين من التركيب و الرقم *3* لتكرار *3* نسخ و هكذا.

تحديد تراكيب الإكساء

يوجد نوعان من تراكيب الإكساء، تراكيب الإكساء أحادية البعد و تراكيب الإكساء ثنائية البعد التي تعتبر الأكثر استخداماً و لنتناول هذين النوعين بشيء من التفصيل:

التراكيب ثنائية البعد

الشكل العام لأمر تعريف التراكيب ثنائية البعد:

*void glTexImage2D(GLenum target, GLint level, GLint components, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *pixels);*

target: يحدد التركيب الهدف . يجب أن يكون GL_TEXTURE_2D .

level : يستخدم عند تطبيق عدة أنواع دقة على التركيب. إذا كان لدينا دقة واحدة فقط تكون قيمتها *0* . مع زيادة القيمة تنقص الدقة.

components : رقم صحيح من ١-٤ يحدد المكونات اللونية في التركيب.

الرقم ١ يحدد مكون اللون الأحمر *R* .

الرقم ٢ يحدد مكون اللون الأحمر *R* و الشفافية *A* .

الرقم ٣ يحدد مكونات الألوان *RGB* .

الرقم ٤ يحدد مكونات الألوان *RGBA* .

width : يحدد عرض صورة التركيب . قيمته يجب أن تكون

2^n+2(border)* حيث *n* عدد صحيح .

height: يحدد ارتفاع صورة التركيب . قيمته يجب أن تكون

2^m+2(border)* حيث *m* عدد صحيح .

border: عرض الاطار . قيمته *0* أو *1* .

format: يصف شكل بيانات صورة التركيب. قيمة كل مكون :

GL_COLOR_INDEX, GL_RGB, GL_RGBA, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_LUMINANCE, or GL_LUMINANCE_ALPHA

type : يصف نوع بيانات صورة التركيب. قيمه

GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT, or GL_BITMAP

pixels : يعين مؤشراً إلى بيانات الصورة في الذاكرة .

التراكيب أحادية البعد

أحياناً يكون الإكساء أحادي البعد كاف (مثلاً إذا رسمت حزم مكسوة بحيث تكون كل الانحرافات في اتجاه واحد)، يتصرف الإكساء أحادي البعد مثل الثنائي البعد ولكن بقيمة *Height = 1* و بدون إطار على طول *Top* و *Bottom*.

الشكل العام لأمر تعريف التراكيب أحادية البعد:

```
void glTexImage1D(GLenum target, GLint level, GLint components, GLsizei width, GLint border, GLenum format, GLenum type, const GLvoid *pixels);
```

كل الوسائط لها نفس المعنى الوارد في ثنائي البعد، باستثناء أن الصورة هنا مصفوفة أحادية البعد و ليست ثنائية البعد.

إسناد إحداثيات التراكيب

يجب عليك عند رسم مشهد فيه إكساء أن تحدد إحداثيات الجسم و إحداثيات التراكيب لكل رأس. تماماً بنفس الطريقة كما تضاف الألوان بين رأسين لمضلع مظلل فإن إحداثيات التراكيب تضاف بشكل خطي بين الرؤوس (تذكر أن التركيب عبارة عن مصفوفة مستطيلة من المعطيات). يمكن أن تتألف إحداثيات الإكساء من واحد أو اثنان أو ثلاثة أو أربع إحداثيات يشار إليها بـــ *s,t,r,q* لتمييزها عن إحداثيات الجسم .*(x,y,z,w)*

في التراكيب أحادية البعد نستخدم الإحداثي *s* .

في التراكيب ثنائية البعد نستخدم الإحداثي *t* و *s* و نتجاهل *r* (ربما تستخدم في المستقبل) و*q* مشابهة لـــ *w* تأخذ القيمة *1* و ممكن أن تستخدم لإنشاء إحداثيات متجانسة.

الأمر الذي يحدد إحداثيات التراكيب مشابه لأمر *Vertex( )* أو *Color( )* حيث يستخدم بنفس الطريقة بين تعليمتي *Begin* و *End*.

تكون هذه القيم(الإحداثيات) عادة ضمن المجال *[1 , 0]* و يمكن أن تحدد خارج هذا المجال. الشكل العام لأمر إحداثيات التراكيب:

```
void glTexCoord{1234}{sifd}{v}(TYPE coords);
```

مع تعليمة ( )glTexCoord1 تأخذ s القيمة المحددة و t,r = 0 ، q = 1

مع تعليمة ( )glTexCoord2 يسمح لك بتحديد t,s و r = 0 ، q = 1

مع تعليمة ( )glTexCoord3 تكون q = 1 وباقي الإحداثيات أنت تحددها.

و يمكن أن تحدد كل الإحداثيات مع ( )glTexCoord4.

استخدم اللاحقة المناسبة (s,i,f,d) و القيمة الموافقة لـ TYPE (قيم TYPE يمكن أن تكون GLshort أو Glint أو GLfloat أو GLdouble) لتحديد نوع معطيات الإحداثيات . يمكن أن تزود الإحداثيات بشكل فردي أو باستخدام المصفوفة (الشعاع) v حيث تعرف إحداثيات أحادية، و بذلك تضرب إحداثيات الإكساء بمصفوفة التراكيب ذات الأبعاد 4×4 قبل أن تحدث عملية الإكساء (لاحظ أن الإحداثيات الصحيحة تفسر مباشرة أفضل من أن تقرب للمجال [1,1-] كالإحداثيات العادية).

تكرار و حصر التراكيب

يمكنك أن تحدد إحداثيات التراكيب خارج المجال [0,1] و أن تحصر أو تكرر خريطة التركيب. في حالة الإكساءات المكررة، إذا كان لديك خريطة كبيرة مع إحداثيات تراكيب تبدأ من 0 إلى 10 في كلا الاتجاهين، يمكنك أن تحصل على 100نسخة من التراكيب المحاذية لبعضها البعض على الشاشة.

تتمثل الإمكانية الأخرى بحصر التراكيب (Clamp)، فأي قيمة أكبر من الواحد تأخذ القيمة 1 و أصغر من الصفر تأخذ صفر، و يكون الحصر مفيداً في التطبيقات التي تريد أن تغطي نسخة وحيدة من التركيب سطحاً كبيراً، فإذا كانت إحداثيات تراكيب السطح ضمن المجال من 0 إلى 10 بكلا الاتجاهين عندئذ تظهر نسخة واحدة من الإكساء في الزاوية السفلية من السطح و باقي السطح يطلى بلون حافة الإكساء حسب الحاجة.

يستخدم التابع glTexParameter() لتعيين بارامترات التركيب ،و له الشكل العام التالي :

```
void glTexParameter{if}{v}(GLenum target, GLenum pname, TYPE
param);
```

target: تأخذ القيمة *GL_TEXTURE_2D* أو *GL_TEXTURE_1D*.

قيم *pname* و *param* مبينة في الجدول التالي:

pname	param
GL_TEXTURE_WRAP_S	GL_CLAMP, GL_REPEAT
GL_TEXTURE_WRAP_T	GL_CLAMP, GL_REPEAT
GL_TEXTURE_MAG_FILTER	GL_NEAREST, GL_LINEAR
GL_TEXTURE_MIN_FILTER	GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_LINEAR
GL_TEXTURE_BORDER_COLOR	any four values in [0, 1]

نلخص فيما يلي خطوات إكساء عناصر مشهد:

١. تحديد التراكيب المراد تطبيقها على العناصر: تحديد ملفات صور التراكيب في الغالب.

٢. توليد التراكيب:

```
Void glGenTextures ( GLsizei n, GLuint *texture );
```

- *n*: عدد أسماء التراكيب المراد توليدها .
- *texture**: مؤشر لأول عنصر في مصفوفة تخزن فيها أسماء التراكيب المولدة .

٣. ربط اسم التركيب المولد مع تركيب هدف (تركيب حقيقي يتم العمل معه)، وتحديد الإكساء الحالي المراد العمل معه:

```
void glBindTexture ( GLenum target, GLuint texture );
```

- *target* : يحدد تركيب الهدف الذي سيربط به اسم التركيب المولد .يجب أن تكون القيمة إما *GL_TEXTURE_1D* أو *GL_TEXTURE_2D*

- *texture* : اسم التركيب الحقيقي المراد التعامل معه .

٤. تحديد صورة تركيب ثنائية البعد:

```
void glTexImage2D(GLenum target,
                  GLint level,
                  GLint components,
                  GLsizei width,
                  GLsizei height,
                  GLint border,
                  GLenum format,
                  GLenum type,
                  const GLvoid *pixels)
```

٥. تعيين بارمترات التركيب

```
void glTexParameter{fi}(GLenum target,GLenum pname,GLfloat
param);
void glTexParameter{fi}v(GLenum target,GLenum pname,GLfloat
*params);
```

٦. تفعيل التركيب

```
glEnable(GL_TEXTURE_1D);
glEnable(GL_TEXTURE_2D);
```

ويلغى تفعيله باستخدام *glDisable()* مع أحد الوسيطين السابقين .

٧. تعيين احداثيات التركيب الحالي

```
void glTexCoord1{dfis}(GLdouble s);
void glTexCoord2{dfis}(GLdouble s,GLdouble t);
void glTexCoord3{dfis}(GLdouble s,GLdouble t,GLdouble r);
void glTexCoord4{dfis}(GLdouble s,GLdouble t,GLdouble r,GLdouble
q);
void glTexCoord1{dfis}v(const GLdouble *v);
void glTexCoord2{dfis}v(const GLdouble *v);
void glTexCoord3{dfis}v(const GLdouble *v);
void glTexCoord4{dfis}v(const GLint *v);
```

حيث *s,t,r,q* تمثل إحداثيات التركيب وهي تكافئ الإحداثيات *x,y,z,w* المستخدمة في العناصر .

تطبيقات عملية

تطبيق *1*

رقعة الشطرنج

١. يعمل هذا التطبيق على رسم رقعة شطرنج . التركيب *Texture* هنا عبارة عن مربعات بيضاء وسوداء متناوبة مولد ضمن نفس البرنامج . يطبق البرنامج هذا التركيب على مربعين (رقعتين من الشطرنج) أحدهما يقابل المشاهد والآخر منحرف عن عين الناظر بزاوية *45* درجة . أنشئ تطبيقاً اسمه *application91* وأنشئ ضمنه الملف *a22.cpp* ،ثم اكتب النص البرمجي التالي ضمن الملف *a22.cpp* :

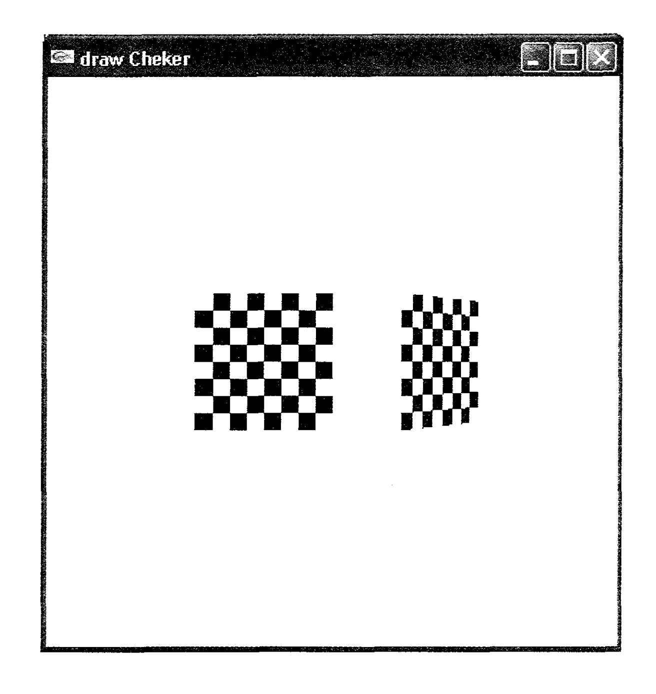
```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>
#define   checkImageWidth 64
#define   checkImageHeight 64
GLubyte checkImage[checkImageWidth][checkImageHeight][3];
    //توليد صورة التركيب
void makeCheckImage(void)
    {
      int i, j, r, c;
        for (i = 0; i < checkImageWidth; i++) {
         for (j = 0; j < checkImageHeight; j++) {
            c = ((((i&0x8)==0)^((j&0x8))==0))*255;
            checkImage[i][j][0] = (GLubyte) c;
            checkImage[i][j][1] = (GLubyte) c;
            checkImage[i][j][2] = (GLubyte) c;
         }
      }
    }

void myinit(void)
```

```
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glEnable(GL_DEPTH_TEST);
   glDepthFunc(GL_LEQUAL);
   makeCheckImage();
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
   glTexImage2D(GL_TEXTURE_2D, 0, 3, checkImageWidth,
   checkImageHeight, 0, GL_RGB, GL_UNSIGNED_BYTE,
  &checkImage[0][0][0]);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
  GL_REPEAT);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
  GL_REPEAT);
  glTexParameterf(GL_TEXTURE_2D,
 GL_TEXTURE_MAG_FILTER, GL_NEAREST);
 glTexParameterf(GL_TEXTURE_2D,
 GL_TEXTURE_MIN_FILTER, GL_NEAREST);
 glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_DECAL);
glEnable(GL_TEXTURE_2D);
glShadeModel(GL_FLAT);
}
static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
 GLUT_DEPTH);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(400,400);
glutCreateWindow("draw Cheker");
 glutDisplayFunc(redraw);
 myinit();
 glMatrixMode(GL_PROJECTION);
 gluPerspective(45,1.0,10.0,200.0);
 glMatrixMode(GL_MODELVIEW);
 glutMainLoop();
 return 0;
}
static void redraw(void)
```

```
{
 glClearColor(1.0,1.0,1.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glTranslatef(0.0f,0.0f,-10.0f);
 glColor3f(0.0, 0.0, 0.0);
 glBegin(GL_QUADS);
     glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
     glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);
     glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 1.0, 0.0);
     glTexCoord2f(1.0, 0.0); glVertex3f(0.0, -1.0, 0.0);
     glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
     glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
     glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
     glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0,-1.41421);
 glEnd();
 glutSwapBuffers();
}
```

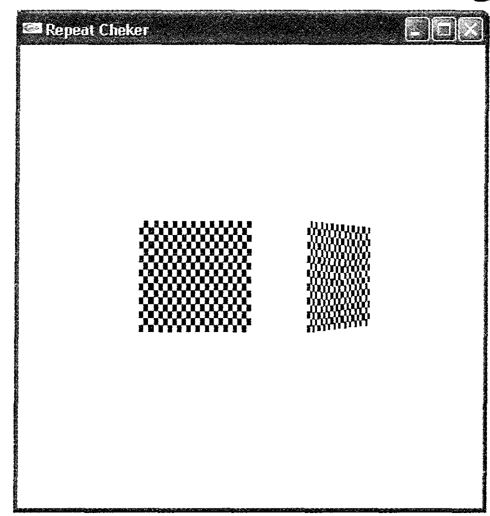
نفذ التطبيق السابق لتشاهد الشكل التالي:



٢. سنعمل الآن على تكرار التركيب السابق *3* مرات وفق المحور *s(x)* و *2* مرة وفق المحور *t(y)*. عدل ضمن التابع *redraw* الذي يصبح كما يلي:

```
static void redraw(void)
     {
     glClearColor(1.0,1.0,1.0,0.0);
     glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
     glLoadIdentity();
     glTranslatef(0.0f,0.0f,-10.0f);
     glColor3f(0.0, 0.0, 0.0);
     glBegin(GL_POLYGON);
       glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
       glTexCoord2f(0.0, 2.0); glVertex3f(-2.0, 1.0, 0.0);
       glTexCoord2f(3.0, 2.0); glVertex3f(0.0, 1.0, 0.0);
       glTexCoord2f(3.0, 0.0); glVertex3f(0.0, -1.0, 0.0);
     glEnd();
     glBegin(GL_POLYGON);
      glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
      glTexCoord2f(0.0, 2.0); glVertex3f(1.0, 1.0, 0.0);
      glTexCoord2f(3.0, 2.0); glVertex3f(2.41421, 1.0, -1.41421);
      glTexCoord2f(3.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
     glEnd();
     glutSwapBuffers();
     }
```

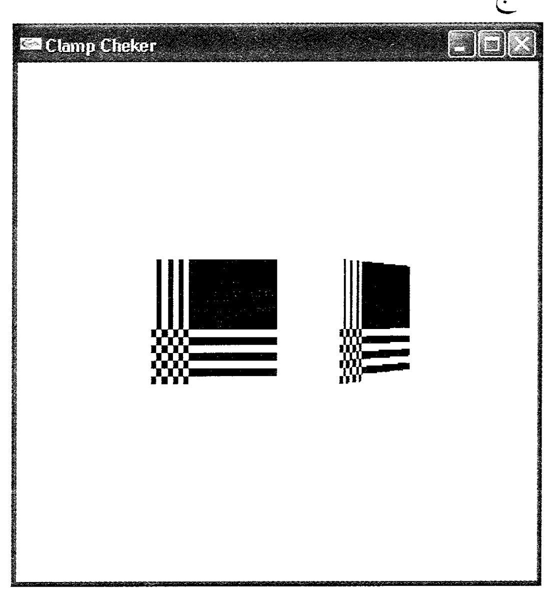

يصبح الشكل الناتج:



٣. سنعمل على استخدام *GL_CLAMP* بدلاً من *GL_REPEAT* ضمن التابع *glTexParameter*()* ، مع ابقاء التابع *redraw* كما في الحالة ٢ السابقة . عدل القيمة ضمن التابع *myinit()* لتصبح :

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP);
 glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP);
```

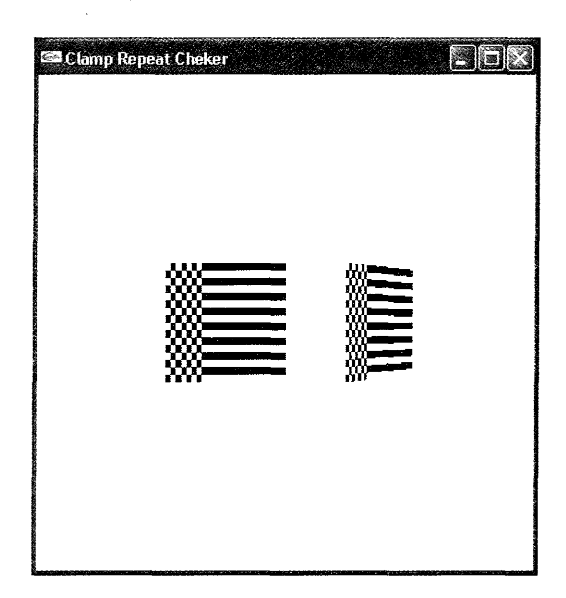

يصبح الشكل الناتج:



٤. يمكن تطبيق *repeat* في اتجاه و *clamp* في اتجاه آخر. أي عدل التابع *myinit()* كما يلي:

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_REPEAT);
```

الشكل الناتج:



تطبيق 2

إكساء مكعب بصورة

يعمل هذا التطبيق على رسم مكعب وإكسائه بصورة شمعة (الملف *app92.bmp*) ، بالإضافة إلى تدوير المكعب بشكل مستمر وكذلك إمكانية التحكم بتدوير المكعب باستخدام الفأرة ، يمكنك أيضاً ضغط المفتاح *f* من لوحة المفاتيح للانتقال إلى نمط ملئ الشاشة أو ضغط المفتاح *w* للعودة إلى الإطار الطبيعي. أنشئ تطبيقاً اسمه *application92* وأنشئ ضمنه الملف *a23.cpp* وتأكد من وضع ملف الصورة ضمن نفس المجلد ،ثم اكتب النص البرمجي التالي ضمن الملف *a23.cpp* :

```
#include <windows.h>
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
GLuinttexture[1];  // يعين فراغ لحفظ تركيب واحد
// زوايا تدوير المكعب
float angle, angle2;
```

```
int moving, startx, starty;
void redraw ( void );
التأكد من وجود ملف الصورة، ثم فتحه وأخذ معلومات منه وتحويله إلى تركيب//
void load_texture ( char *file_name, int width, int height, int depth,
                    GLenum colour_type, GLenum filter_type )
    {
      GLubyte *raw_bitmap ;
      FILE *file;
      if ((file = fopen(file_name, "rb"))==NULL )
      {
        printf ( "File Not Found : %s\n", file_name );
        exit   ( 1 );
      }
      raw_bitmap = (GLubyte *) malloc ( width * height * depth *
      ( sizeof(GLubyte)) );
      if (raw_bitmap == NULL)
      {
        printf ( "Cannot allocate memory for texture\n" );
        fclose ( file);
        exit   ( 1 );
      }
      fread  ( raw_bitmap , width * height * depth, 1 , file );
      fclose ( file);
      // تحديد نمط التصفية
      glTexParameteri ( GL_TEXTURE_2D,
      GL_TEXTURE_MAG_FILTER, filter_type );
      glTexParameteri ( GL_TEXTURE_2D,
      GL_TEXTURE_MIN_FILTER, filter_type );
      // تحديد بيئة التركيب
      glTexEnvf ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
     GL_MODULATE );
      // بناء التركيب
      gluBuild2DMipmaps ( GL_TEXTURE_2D, colour_type, width,
      height, colour_type, GL_UNSIGNED_BYTE, raw_bitmap);
      // تحرير المصفوفة
      free ( raw_bitmap );
    }
```

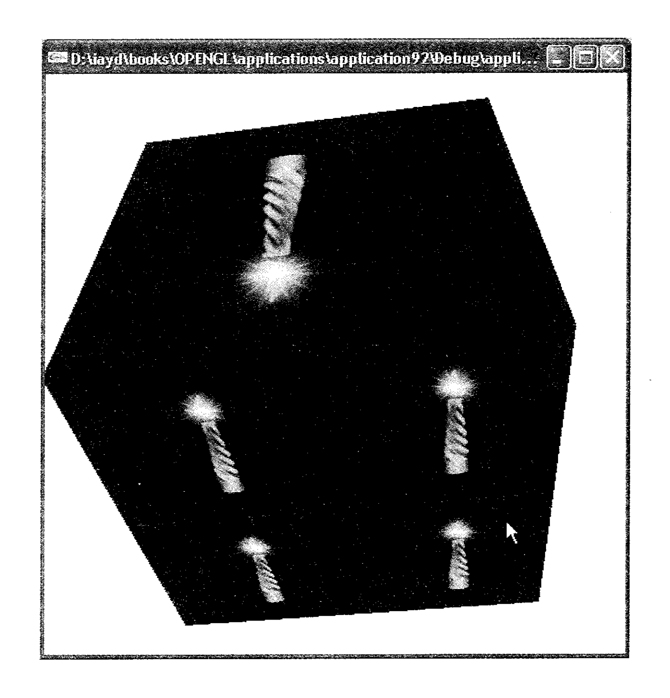
```
void init ( void )
    {
      glEnable ( GL_TEXTURE_2D );
      glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );
      load_texture ( "app92.bmp", 256, 256, 3, GL_RGB,
     GL_LINEAR );
      // Leave depth test off for speed, theres no 'in and out'
     movement anyhoo :)
      // glEnable (GL_DEPTH_TEST);
      glEnable ( GL_CULL_FACE );
      glClearColor (1.0, 1.0, 1.0, 0.0);
    }
void draw_box ( void )
    {
      glPushMatrix ( );
      glRotatef ( 90.0, 1.0, 0.0, 0.0 );
      glColor4f ( 1.0, 1.0, 1.0, 1.0  );
      glBindTexture(GL_TEXTURE_2D, texture[0]);
     glBegin(GL_QUADS);
            // Front Face
            glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
            glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
            glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
            glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);
            // Back Face
            glTexCoord2f(2.0f, 0.0f); glVertex3f(-1.0f, -1.0f,-1.0f);
            glTexCoord2f(2.0f, 2.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
            glTexCoord2f(0.0f, 2.0f); glVertex3f( 1.0f,  1.0f, -1.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
            // Top Face
            glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f,  1.0f,  1.0f);
            glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
            glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);
            // Bottom Face
            glTexCoord2f(1.0f, 1.0f);glVertex3f(-1.0f, -1.0f, -1.0f);
            glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
            glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
            // Right face
```

```
            glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
            glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);
            glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
            // Left Face
            glTexCoord2f(0.0f, 0.0f);glVertex3f(-1.0f, -1.0f, -1.0f);
            glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
            glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);
            glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
       glEnd();
       glPopMatrix ( );
     }
void reshape ( int w, int h )
     {
       glViewport   ( 0, 0, (GLint) w, (GLint) h );
       glMatrixMode ( GL_PROJECTION);
       glLoadIdentity( );
       if ( h==0 )
         gluPerspective ( 45, (GLdouble)w, 1.0, 2000.0 );
       else
         gluPerspective(45,(GLdouble)w/(GLdouble)h,1.0, 2000.0 );
       glMatrixMode ( GL_MODELVIEW );
       glLoadIdentity ( );
     }
void mouse ( int button, int state, int x, int y )
     {
       if ( button==GLUT_LEFT_BUTTON && state==GLUT_DOWN )
       {
         moving = 1;
         startx = x;
         starty = y;
       }
       if ( button == GLUT_LEFT_BUTTON && state == GLUT_UP  )
         moving = 0;
     }
void motion ( int x, int y )
     {
       if ( moving )
       {
         angle = angle +   ( x - startx );
         angle2 = angle2 + ( y - starty );
```

```
            startx = x;
            starty = y;
            glutPostRedisplay ( );
          }
        }
void idle_func ( void )
        {
          angle = angle + 0.1;
          if ( angle == 360 )
            angle = 0;
          glutPostRedisplay ( );
        }
void keyboard ( unsigned char key, int x, int y )
        {
          switch ( key )
          {
            case 27:  /*  Escape key  */
              exit(0);
              break;
            case 'f':
             glutFullScreen ( );
             break;
            case 'w':
              glutReshapeWindow ( 500,500 );
              break;
            default:
              break;
          }
        }
/*  Main Loop
 *  Open window with initial window size, title bar,
 *  RGBA display mode, and handle input events. */
int main ( int argc, char** argv )
        {
          glutInit ( &argc, argv );
          glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE );
          glutInitWindowSize ( 250, 250);
          glutCreateWindow ( argv[0] );
          init();
          glutReshapeFunc  ( reshape   );
          glutMouseFunc    ( mouse     );
```

```
    glutMotionFunc   ( motion    );
    glutKeyboardFunc ( keyboard  );
    glutDisplayFunc  ( redraw    );
    glutIdleFunc     ( idle_func );
    glutMainLoop ( );
    return 0;
  }
void redraw ( void )
  {
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix ( );
    glTranslatef ( 0.0, 0.0, -4.0 );
    glRotatef    ( angle2, 1.0, 0.0, 0.0 );
    glRotatef    ( angle, 0.0, 1.0, 0.0  );
    draw_box   ( );
    glPopMatrix  ( );
    glutSwapBuffers( );
  }
```


نفذ التطبيق السابق لتشاهد الشكل التالي:



تطبيق 3

إكساء المكعبات المتراكبة المدروسة سابقاً التطبيق 4 من الفصل السادس

يعمل هذا التطبيق على رسم *15* مكعب متركبة وإكسائها بصورة جميلة (الملف *app93.bmp*).أنشئ تطبيقاً اسمه *application93* وأنشئ ضمنه الملف *a24.cpp* وتأكد من وضع ملف الصورة ضمن نفس المجلد ،ثم اكتب النص البرمجي التالي ضمن الملف : *a24.cpp*

```
#include <windows.h>
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
GLuinttexture[1];  // Storage For One Texture ( NEW )
GLuint box,top,xloop,yloop;
GLfloat xrot,yrot;
static GLfloat boxcol[5][3]=
      {
       {1.0f,0.0f,0.0f},{1.0f,0.5f,0.0f},{1.0f,1.0f,0.0f},{0.0f,1.0f,
        0.0f},{0.0f,1.0f,1.0f}
      };
static GLfloat topcol[5][3]=
      {
       {.5f,0.0f,0.0f},{0.5f,0.25f,0.0f},{0.5f,0.5f,0.0f},{0.0f,0.5f,
        0.0f},{0.0f,0.5f,0.5f}
      };
GLvoid BuildLists();
static void redraw(void);
int main(int argc, char **argv);
void load_texture ( char *file_name, int width, int height, int depth,
                    GLenum colour_type, GLenum filter_type )
     {
       GLubyte *raw_bitmap ;
       FILE *file;
       if ((file = fopen(file_name, "rb"))==NULL )
       {
         printf ( "File Not Found : %s\n", file_name );
         exit   ( 1 );
```

```
        }
        raw_bitmap = (GLubyte *) malloc ( width * height * depth *
       ( sizeof(GLubyte)) );
        if (raw_bitmap == NULL)
        {
          printf ( "Cannot allocate memory for texture\n" );
          fclose ( file);
          exit   ( 1 );
        }
        fread  ( raw_bitmap , width * height * depth, 1 , file );
        fclose ( file);
        //  Set Filtering type
        glTexParameteri ( GL_TEXTURE_2D,
        GL_TEXTURE_MAG_FILTER, filter_type );
        glTexParameteri ( GL_TEXTURE_2D,
        GL_TEXTURE_MIN_FILTER, filter_type );
        //  Set Texture Evironment

        glTexEnvf ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
        GL_MODULATE );
        //  Build Mipmaps
        gluBuild2DMipmaps ( GL_TEXTURE_2D, colour_type, width,
        height,  colour_type, GL_UNSIGNED_BYTE, raw_bitmap);
        //  Free up the array
        free ( raw_bitmap );
     }
void init ( void )
     {
         glEnable ( GL_TEXTURE_2D );
         glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );
         load_texture ( "app93.bmp", 128, 128, 3, GL_RGB,
         GL_LINEAR );
         glEnable ( GL_CULL_FACE );
        glClearColor (1.0, 1.0, 1.0, 0.0);
     }
GLvoid BuildLists()
     {
       box=glGenLists(2);
       glNewList(box,GL_COMPILE);
       glBegin(GL_QUADS);
              // Bottom Face
```

```
            glNormal3f( 0.0f,-1.0f, 0.0f);
            glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,-1.0f, -1.0f);
            glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
            glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
            // Front Face
            glNormal3f( 0.0f, 0.0f, 1.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
            glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
            glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
            glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);
            // Back Face
            glNormal3f( 0.0f, 0.0f,-1.0f);
            glTexCoord2f(1.0f,0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
            glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
            glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
            // Right face
            glNormal3f( 1.0f, 0.0f, 0.0f);
            glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
            glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);
            glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
            // Left Face
            glNormal3f(-1.0f, 0.0f, 0.0f);
            glTexCoord2f(0.0f,0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
            glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
            glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);
            glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
      glEnd();
    glEndList();
    top=box+1;
    glNewList(top,GL_COMPILE);
       glBegin(GL_QUADS);
            // Top Face
            glNormal3f( 0.0f, 1.0f, 0.0f);
            glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f,  1.0f,  1.0f);
            glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
            glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);
       glEnd();
```

```
    glEndList();
   }
int main(int argc, char **argv)
   {
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
    GLUT_DEPTH);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(400,400);
    glutCreateWindow("draw 15 Lighting&Texturing Boxes");
    glutDisplayFunc(redraw);
    BuildLists();
    init();
    glMatrixMode(GL_PROJECTION);
    gluPerspective(45,1.0,10.0,200.0);
    glMatrixMode(GL_MODELVIEW);
    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHTING);
    glEnable(GL_COLOR_MATERIAL);
    glutMainLoop();
    return 0;
   }
static void redraw(void)
   {
    glClearColor(1.0,1.0,1.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    for (yloop=1;yloop<6;yloop++)
    {
        for (xloop=0;xloop<yloop;xloop++)
        {
            glLoadIdentity();
            glTranslatef(1.4f+(float(xloop)*2.8f)-
            (float(yloop)*1.4f),((6.0f-float(yloop))*2.4f)-
            7.0f,-20.0f);
            glRotatef(45.0f-
           (2.0f*yloop)+xrot,1.0f,0.0f,0.0f);
            glRotatef(45.0f+yrot,0.0f,1.0f,0.0f);
            glColor3fv(boxcol[yloop-1]);
            glCallList(box);
            glColor3fv(topcol[yloop-1]);
            glCallList(top);
```

```
            }
        }


        glutSwapBuffers();
    }
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



تطبيق 4
الكرة الأرضية

يعمل هذا التطبيق على رسم كرة وإكسائها بخريطة الكرة الأرضية (الملف *earth_r.raw*) ، بالإضافة إلى تدوير الكرة الأرضية بشكل مستمر وكذلك إمكانية التحكم بتدوير الكرة الأرضية باستخدام الفأرة ، يمكنك أيضاً ضغط المفتاح *f* من لوحة المفاتيح للانتقال إلى نمط ملئ الشاشة أو ضغط المفتاح *w* للعودة إلى الإطار الطبيعي. أنشئ تطبيقاً اسمه *application94* وأنشئ ضمنه الملف *a25.cpp* وتأكد من وضع ملف الصورة ضمن نفس المجلد ،ثم اكتب النص البرمجي التالي ضمن الملف *a25.cpp* :

```
#include <windows.h>
#include <GL/glut.h>
```

```
#include <stdio.h>
#include <stdlib.h>
// Angles here
float angle, angle2;
int moving, startx, starty;
GLfloat  ambient_light[] = {0.3, 0.3, 0.45, 1.0};
GLfloat   source_light[] = {0.9, 0.8, 0.8, 1.0};
GLfloat      light_pos[] = {7.0, 0.0, 0.0, 1.0};
void load_texture ( char *file_name, int width, int height, int depth,
                    GLenum colour_type, GLenum filter_type )
    {
      GLubyte *raw_bitmap ;
      FILE *file;
      if ((file = fopen(file_name, "rb"))==NULL )
      {
        printf ( "File Not Found : %s\n", file_name );
        exit   ( 1 );
      }
      raw_bitmap = (GLubyte *) malloc ( width * height * depth *
      ( sizeof(GLubyte)) );
      if (raw_bitmap == NULL)
      {
        printf ( "Cannot allocate memory for texture\n" );
        fclose ( file);
        exit   ( 1 );
      }
      fread  ( raw_bitmap , width * height * depth, 1 , file );
      fclose ( file);
      //  Set Filtering type
      glTexParameter(GL_TEXTURE_2D,
     GL_TEXTURE_MAG_FILTER, filter_type );
     glTexParameteri ( GL_TEXTURE_2D,
    GL_TEXTURE_MIN_FILTER, filter_type );
     //  Set Texture Evironment
     glTexEnvf ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
    GL_MODULATE );
     //  Build Mipmaps
     gluBuild2DMipmaps ( GL_TEXTURE_2D, colour_type, width,
    height,  colour_type, GL_UNSIGNED_BYTE, raw_bitmap );
      //  Free up the array
      free ( raw_bitmap );
```

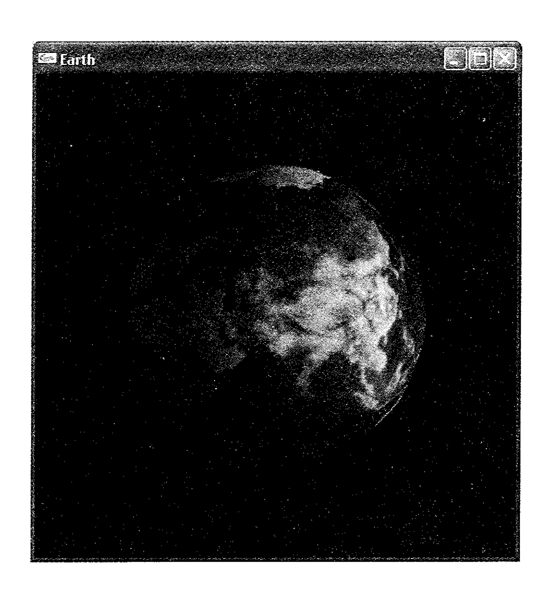
```
    }
void init ( void )
    {
      // Set up simple lighting model
      glEnable ( GL_LIGHTING );
      glLightModelfv(GL_LIGHT_MODEL_AMBIENT,ambient_light );
      glLightfv (GL_LIGHT0,GL_DIFFUSE, source_light );
      glLightfv ( GL_LIGHT0,GL_POSITION, light_pos  );
      glEnable  ( GL_LIGHT0 );
      // Enable material properties for lighting
      glEnable ( GL_COLOR_MATERIAL );
      glColorMaterial ( GL_FRONT, GL_AMBIENT_AND_DIFFUSE );
      glEnable ( GL_TEXTURE_2D );
      glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );
      load_texture("earth_r.raw",640,320,3,GL_RGB,GL_LINEAR );
      glEnable ( GL_CULL_FACE );
      glClearColor (0.0, 0.0, 0.0, 0.0);
    }
void draw_earth ( void )
    {
      glPushMatrix ( );
      glRotatef ( 90.0, 1.0, 0.0, 0.0 );
      glColor4f ( 1.0, 1.0, 1.0, 1.0  );
      GLUquadricObj*  q = gluNewQuadric ( );
      gluQuadricDrawStyle ( q, GLU_FILL );
      gluQuadricNormals ( q, GLU_SMOOTH );
      gluQuadricTexture ( q, GL_TRUE    );
      gluSphere ( q, 1.0, 20, 20 );
      gluDeleteQuadric ( q );
      glPopMatrix ( );
    }
void redraw ( void )
    {
      glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      glLightfv ( GL_LIGHT0,GL_POSITION, light_pos );
      glPushMatrix ( );
      glTranslatef ( 0.0, 0.0, -4.0 );
      glRotatef   ( angle2, 1.0, 0.0, 0.0 );
      glRotatef   ( angle, 0.0, 1.0, 0.0  );
      draw_earth   ( );
      glPopMatrix  ( );
```

```
        glutSwapBuffers( );
    }
void reshape ( int w, int h )
    {
        glViewport   ( 0, 0, (GLint) w, (GLint) h );
        glMatrixMode ( GL_PROJECTION);
        glLoadIdentity( );
        if ( h==0 )
          gluPerspective ( 45, (GLdouble)w, 1.0, 2000.0 );
        else
          gluPerspective(45,(GLdouble)w/(GLdouble)h,1.0, 2000.0 );
        glMatrixMode ( GL_MODELVIEW );
        glLoadIdentity ( );
    }
void mouse ( int button, int state, int x, int y )
    {
        if ( button == GLUT_LEFT_BUTTON && state ==
       GLUT_DOWN )
        {
          moving = 1;
          startx = x;
          starty = y;
        }
        if ( button == GLUT_LEFT_BUTTON && state == GLUT_UP  )
          moving = 0;
    }
void motion ( int x, int y )
    {
        if ( moving )
        {
          angle = angle +   ( x - startx );
          angle2 = angle2 + ( y - starty );
          startx = x;
          starty = y;
          glutPostRedisplay ( );
        }
    }
void idle_func ( void )
    {
        angle = angle + 0.1;
        if ( angle == 360 )
```

```
        angle = 0;
      glutPostRedisplay ( );
    }
#pragma argsused
void keyboard ( unsigned char key, int x, int y )
    {
      switch ( key )
      {
        case 27:  /*  Escape key  */
          exit(0);
          break;
        case 'f':
         glutFullScreen ( );
         break;
        case 'w':
          glutReshapeWindow ( 500,500 );
          break;
        default:
          break;
      }
    }

int main ( int argc, char** argv )
    {
      glutInit ( &argc, argv );
      glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE );
      glutInitWindowSize ( 450, 450);
      glutCreateWindow("Earth");
       init();
      glutReshapeFunc  ( reshape   );
      glutMouseFunc    ( mouse     );
      glutMotionFunc   ( motion    );
      glutKeyboardFunc ( keyboard  );
      glutDisplayFunc  ( redraw   );
      glutIdleFunc     ( idle_func );
      glutMainLoop ( );
      return 0;
    }
```

نفذ التطبيق السابق لتشاهد الشكل التالي:



الملحق آ

المكتبة GL

تعتبر*GL* المكتبة الرئيسية لـ *OpenGL*،وتحوي معظم أوامر*OpenGL* . سنتناول الآن أوامر هذه المكتبة الهامة وسنعمل على تقسيمها إلى مجموعات حسب وظيفتها ، وسنكتفي بشرح الأوامر الجديدة التي لم ترد في فصول الكتاب :

مجموعة العناصر الهندسية Primitives

1. glBegin, glEnd, glVertex, glRect

تستخدم هذه الأوامر لتحديد نقاط (رؤوس) أو مستطيلات .

الشكل العام لهذه الأوامر:

```
void glBegin (GLenum mode);
void glEnd (void);
void glVertex2{sifd}{v} (TYPE x, TYPE y);
void glVertex3{sifd}{v} (TYPE x, TYPE y, TYPE z);
void glVertex4{sifd}{v} (TYPE x, TYPE y, TYPE z, TYPE w);
void glRect{sifd} (TYPE x1, TYPE y1, TYPE x2, TYPE y2);
void glRect{sifd}v (const TYPE *v1, const TYPE *v2);
```

الأوامر السابقة مشروحة بالتفصيل ضمن فصول الكتاب.

2. glEdgeFlag, glEdgeFlagv

يستخدم هذان الأمران لتحديد طريقة معاملة حواف المضلعات .

الشكل العام لهذين الأمرين:

```
void glEdgeFlag (GLboolean flag);
```

يعلم حواف المضلعات والرباعيات والمثلثات إما بشكل متجاور *boundary* أو غير متجاور *nonboundary*.

flag: يعين القيمة الحالية لعلم الحافة . قيمته إما *true* أو *false* . قيم رؤوس الرباعيات والمثلثات المتصلة تكون *boundary* بغض النظر عن قيمة علم الحافة .

```
void glEdgeFlagv (const GLboolean *flag);
```

نفس الأمر السابق ولكن باستخدام الشعاع (المصفوفة) .

مجموعة تحويل الإحداثيات Coordinate Transformation

1. glRotate, glTranslate, glScale, glMultMatrix, glFrustum, glOrtho

تستخدم هذه الأوامر لتطبيق تحويل على المصفوفة الحالية .

الشكل العام لهذه الأوامر:

```
void glRotate{fd} (TYPE angle, TYPE x, TYPE y, TYPE z);
void glTranslate{fd} (TYPE x, TYPE y, TYPE z);
```

void glScale{fd} (TYPE x, TYPE y, TYPE z);
*void glMultMatrix{fd} (const TYPE *m);*

يضرب المصفوفة الحالية بمصفوفة تحكم .

m : يحدد مؤشراً إلى مصفوفة أبعادها *4X4* مخزنة على شكل عمود من القيم المتتالية.

void glFrustum (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);

يضرب المصفوفة الحالية بالمصفوفة المنظورية *perspective* (تحويل إسقاط) .

left, right: يحددان إحداثيات مستويات القطع الشاقولية اليساري واليميني .

bottom, top: يحددان إحداثيات مستويات القطع الأفقية العلوي والسفلي .

near, far: يحددان المسافة القريبة والبعيدة لعمق مستويات القطع . والقيم هنا يجب أن تكون موجبة .

void glOrtho (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);

2. glLoadMatrix, glLoadIdentity

يستخدم هذان الأمران لاستبدال المصفوفة الحالية .

الشكل العام لهذين الأمرين:

*void glLoadMatrix{fd} (const TYPE *m);*

يستبدل المصفوفة الحالية بمصفوفة تحكم .

m : يحدد مؤشراً إلى مصفوفة أبعادها *4X4* مخزنة على شكل عمود من القيم المتتالية.

void glLoadIdentity (void);

3. glMatrixMode, glPushMatrix, glPopMatrix

تستخدم هذه الأوامر للتعامل مع مصفوفة المكدس .

الشكل العام لهذه الأوامر:

void glMatrixMode (GLenum mode);

يحدد أي مصفوفة ستكون المصفوفة الحالية .

mode: يحدد المصفوفة التي ستكون المصفوفة الحالية . قيم هذا الوسيط :

GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE.
void glPushMatrix (void);
void glPopMatrix (void);

4. glDepthRange, glViewport

يستخدم هذان الأمران لتحديد تحويل *viewport*.

الشكل العام لهذين الأمرين:

void glDepthRange (GLclampd near, GLclampd far);

يحدد كيفية تنظيم قيم *z* من إحداثيات العنصر إلى إحداثيات الإطار .

near: يحدد تنظيم مستوي القطع القريب إلى إحداثيات الإطار . القيمة الافتراضية *0*

far: يحدد تنظيم مستوي القطع البعيد إلى إحداثيات الإطار . القيمة الافتراضية *1* .

void glViewport (GLint x, GLint y, GLsizei width, GLsizei height);

مجموعة الألوان والإضاءة *Coloring and Lighting*

1. glColor, glIndex, glNormal3

تستخدم هذه الأوامر لتطبيق تحويل على المصفوفة الحالية .

الشكل العام لهذه الأوامر:

void glColor3{bsifd ubusui}{v} (TYPE red, TYPE green, TYPE blue);
void glColor4{bsifd ubusui}{v} (TYPE red, TYPE green, TYPE blue, TYPE alpha);
void glIndex{sifd}{v} (TYPE index);
void glNormal3{bsifd}{v} (TYPE nx, TYPE ny, TYPE nz);

2. glLight, glMaterial, glLightModel

تستخدم هذه الأوامر لتحديد وسائط مصدر الضوء أو المادة أو أنماط الضوء.

الشكل العام لهذه الأوامر:

void glLight{if}{v} (GLenum light, GLenum pname, TYPE param);
void glMaterial{if}{v} (GLenum face, GLenum pname, TYPE param);
void glLightModel{if}{v} (GLenum pname, TYPE param);

3. glShadeModel

يستخدم هذا الأمر لتحديد نمط التظليل .

الشكل العام لهذا الأمر:

void glShadeModel (GLenum mode);

4. glFrontFace

يستخدم هذا الأمر لتحديد اتجاه المضلع الذي سيعتبر الوجه الأمامي .

الشكل العام لهذا الأمر:

```
void glFrontFace (GLenum dir);
```

يعرف اتجاه المضلع الذي سيعتبر الوجه الأمامي.

mode: يعرف اتجاه مضلع الوجه الأمامي . قيمه إما مع عقارب الساعة *GL_CW* (القيمة الافتراضية) أو عكس عقارب الساعة *GL_CCW* .

5. glColorMaterial

يتسبب هذا الأمر بجعل لون المادة يتعلق باللون الحالي.

الشكل العام لهذا الأمر:

```
void glColorMaterial (GLenum face, GLenum mode);
```

6. glGetLight, glGetMaterial

يستخدم هذان الأمران للحصول على مصدر الضوء أو مادة العنصر المضاء.

الشكل العام لهذا الأمر:

```
void glGetLight{if}v (GLenum light, GLenum pname, TYPE
*params);
```

يعيد قيم وسائط مصدر الضوء *light*.

light: يحدد مصدر الضوء .

pname: يحدد وسائط مصدر الضوء . القيم الممكنة :

GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_POSITION, GL_SPOT_DIRECTION, GL_SPOT_EXPONENT, GL_SPOT_CUTOFF, GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION , GL_QUADRATIC_ATTENUATION.

params: يعيد البيانات المطلوبة .

```
void glGetMaterial{if}v (GLenum face, GLenum pname, TYPE
*params);
```

يعيد وسائط مادة معينة .

face: يحدد هل سيتم الاستعلام عن مادة الوجه الأمامي *GL_FRONT* أم مادة الوجه الخلفي *GL_BACK* .

pname: يحدد وسيط المادة المراد إعادة قيمته . القيم المحتملة :

GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION, GL_SHININESS, GL_COLOR_INDEXES.

params: يعيد البيانات المطلوبة .

مجموعة القطع *Clipping*

1. glClipPlane

يستخدم هذا الأمر لتحديد مستوي قطع .

الشكل العام لهذا الأمر:

```
void glClipPlane (GLenum plane, const GLdouble *equation);
```

2. glGetClipPlane

يستخدم هذا الأمر لإعادة معاملات مستوي قطع معين .

الشكل العام لهذا الأمر:

```
void glGetClipPlane (GLenum plane, GLdouble *equation);
```

يعيد معاملات مستوي قطع معين .

plane: يحدد مستوي قطع .

equation: يعيد أربعة أعداد فاصلة عائمة مضاعفة الدقة تمثل معاملات معادلة القطع للمستوي *plane* وفق إحداثيات عين الناظر .

مجموعة خطوط المسح *Rasterization*

1. glRasterPos2

يستخدم هذا الأمر لتعيين موقع خط المسح الحالي .

الشكل العام لهذا الأمر:

```
void glRasterPos2{sifd}{v}(TYPE x, TYPE y);
void glRasterPos3{sifd}{v}(TYPE x, TYPE y, TYPE z);
void glRasterPos4{sifd}{v}(TYPE x, TYPE y, TYPE z, TYPE w);
```

يحدد موقع خط المسح الحالي من أجل عمليات النقاط الضوئية .

x, y, z, w : تعين إحداثيات *x, y, z, w* لموقع خط المسح وفق إحداثيات العنصر .

2. glBitmap

يستخدم هذا الأمر لتعيين صورة نقطية *Bitmap* .

الشكل العام لهذا الأمر:

*void glBitmap (GLsizei width, GLsizei height, GLfloat xorig, GLfloat yorig, GLfloat xmove, GLfloat ymove, const GLubyte *bitmap);*

يرسم صورة نقطية .

width, height: يحددان عرض وارتفاع الصورة النقطية (واحدة القياس هي النقطة الضوئية *pixel*).

xorig, yorig : يحدد موقع بدء الصورة النقطية . يقاس موقع البدء من الزاوية اليسارية السفلية للصورة النقطية .

xmove, ymove : يحدد مقدار الإزاحة *x,y* الواجب إضافتها إلى موقع المسح الحالي بعد رسم الصورة .

bitmap : يحدد عنوان الصورة النقطية .

3. glPointSize, glLineWidth

يستخدم هذان الأمران لتعيين أبعاد النقاط أو الخطوط .

الشكل العام لهذين الأمرين:

void glPointSize (GLfloat size);
void glLineWidth (GLfloat width);

4. glLineStipple, glPolygonStipple, glGetPolygonStipple

تستخدم هذه الأوامر لتحديد نمط خطوط أو نموذج ملء مضلعات أو لإعادة نموذج ملء مضلع معين .

الشكل العام لهذه الأوامر:

void glLineStipple (GLint factor, GLushort pattern);
*void glPolygonStipple (const GLubyte *mask);*
*void glGetPolygonStipple (GLubyte *mask);*

يعيد نموذج ملء المضلعات الحالي .

mask: يعيد نموذج الملء الحالي .

5. glCullFace, glPolygonMode

يستخدم هذان الأمران لانتقاء كيفية المسح الخطي للمضلعات.

الشكل العام لهذين الأمرين:

void glCullFace (GLenum mode);

يحدد فيما إذا كان سيتم اختيار الوجه الأمامي أم الخلفي للعنصر .

mode: يحدد الوجه المرشح لعملية الانتخاب . قيمه *GL_FRONT* و *GL_BACK*.

void glPolygonMode (GLenum face, GLenum mode);

مجموعة عمليات النقاط الضوئية *Pixel Operations*

1. glReadBuffer

يستخدم هذا الأمر لتحديد مصدر قراءة أو نسخ النقاط الضوئية .

الشكل العام لهذا الأمر:

void glReadBuffer (GLenum mode);

يختار مصدر الذاكرة المؤقتة للون من أجل النقاط الضوئية.

mode: يحدد ذاكرة مؤقتة للون . قيمه:

GL_FRONT_LEFT, GL_FRONT_RIGHT, GL_BACK_LEFT, GL_BACK_RIGHT, GL_FRONT, GL_BACK, GL_LEFT, GL_RIGHT, GL_AUXi.

قيمة*i* بين *0* و *GL_AUX_BUFFERS -1* .

2. glReadPixels, glDrawPixels, glCopyPixels

تستخدم هذه الأوامر لقراءة و كتابة و نسخ النقاط الضوئية .

الشكل العام لهذه الأوامر:

*void glReadPixels (GLint x, GLint y, GLsizei width, GLsizei height, GLenum format, GLenum type, GLvoid *pixels);*
*void glDrawPixels (GLsizei width, GLsizei height, GLenum format, GLenum type, const GLvoid *pixels);*
void glCopyPixels (GLint x, GLint y, GLsizei width, GLsizei height, GLenum type);

3. glPixelStore, glPixelTransfer, glPixelMap, glGetPixelMap

تستخدم هذه الأوامر لتحديد كيفية تشفير و معالجة النقاط الضوئية أو للاستعلام عن ذلك .

الشكل العام لهذه الأوامر:

void glPixelStore{if} (GLenum pname, TYPE param);

يعين أنماط تخزين النقاط الضوئية .

pname: يحدد اسم الوسيط المراد تعيين قيمة له .هناك ست قيم تؤثر على رزم بيانات النقاط الضوئية في الذاكرة :

GL_PACK_SWAP_BYTES, GL_PACK_LSB_FIRST, GL_PACK_ROW_LENGTH, GL_PACK_SKIP_PIXELS, GL_PACK_SKIP_ROWS, GL_PACK_ALIGNMENT.

هناك ست قيم أخرى تؤثر على فك رزم بيانات النقاط الضوئية من الذاكرة:

GL_UNPACK_SWAP_BYTES, GL_UNPACK_LSB_FIRST, GL_UNPACK_ROW_LENGTH, GL_UNPACK_SKIP_PIXELS, GL_UNPACK_SKIP_ROWS,GL_UNPACK_ALIGNMENT.

param: يعين قيمة للوسيط المعين في *pname* .

void glPixelTransfer{if} (GLenum pname, TYPE param);

يعين أنماط نقل النقاط الضوئية .

pname: يحدد اسم وسيط نقل النقاط الضوئية المراد تعيين قيمة له . قيمه :

GL_MAP_COLOR, GL_MAP_STENCIL, GL_INDEX_SHIFT, GL_INDEX_OFFSET, GL_RED_SCALE, GL_RED_BIAS, GL_GREEN_SCALE, GL_GREEN_BIAS, GL_BLUE_SCALE, GL_BLUE_BIAS, GL_ALPHA_SCALE, GL_ALPHA_BIAS, GL_DEPTH_SCALE, GL_DEPTH_BIAS.

param: يعين قيمة للوسيط المعين في *pname* .

*void glPixelMap{f usui}v (GLenum map, GLint mapsize, const TYPE *values);*

يجهز خريطة نقل النقاط الضوئية .

map:يحدد اسم خريطة النقل . القيم المحتملة لهذا الخيار :

GL_PIXEL_MAP_I_TO_I, GL_PIXEL_MAP_S_TO_S, GL_PIXEL_MAP_I_TO_R, GL_PIXEL_MAP_I_TO_G,

GL_PIXEL_MAP_I_TO_B, GL_PIXEL_MAP_I_TO_A, GL_PIXEL_MAP_R_TO_R, GL_PIXEL_MAP_G_TO_G, GL_PIXEL_MAP_B_TO_B, GL_PIXEL_MAP_A_TO_A.

mapsize: يحدد حجم الخريطة المعرفة مسبقاً .

values:يعين مصفوفة بقيم *mapsize* .

*void glGetPixelMap{f usui}v (GLenum map, TYPE *values);*

يعيد خريطة النقاط الضوئية المحددة .

map: يحدد خريطة النقاط الضوئية المراد إعادتها . القيم المحتملة لهذا الوسيط :

GL_PIXEL_MAP_I_TO_I, GL_PIXEL_MAP_S_TO_S, GL_PIXEL_MAP_I_TO_R, GL_PIXEL_MAP_I_TO_G, GL_PIXEL_MAP_I_TO_B, GL_PIXEL_MAP_I_TO_A, GL_PIXEL_MAP_R_TO_R, GL_PIXEL_MAP_G_TO_G, GL_PIXEL_MAP_B_TO_B, GL_PIXEL_MAP_A_TO_A.

values:يعيد مصفوفة بمحتويات خريطة النقاط الضوئية .

4. glPixelZoom

يستخدم هذا الأمر لتكبير أو تصغير صور النقاط الضوئية .

الشكل العام لهذا الأمر:

void glPixelZoom (GLfloat xfactor, GLfloat yfactor);

مجموعة الإكساء بالتراكيب

1. glTexParameter, glTexEnv

يستخدم هذان الأمران للتحكم بكيفية تطبيق التركيب على مرحلة الجزء *Fragment*

الشكل العام لهذين الأمرين:

void glTexParameter{if}{v} (GLenum target, GLenum pname, TYPE param);
void glTexEnv{if}{v} (GLenum target, GLenum pname, TYPE param);

يعين وسائط بيئة التركيب .

target: يعين بيئة تركيب . قيمته يجب أن تكون *GL_TEXTURE_ENV* .

pname: يعين تسمية لوسيط بيئة تركيب. قيمته يجب ان تكون *GL_TEXTURE_ENV_MODE* ، ويمكن استخدام قيمة ثانية في حالة استخدام طريقة الشعاع *v* وهي : *GL_TEXTURE_ENV_COLOR*.

param: يحدد إحدى القيم التالية : *GL_MODULATE, GL_DECAL, GL_BLEND*.

2. glTexCoord

يستخدم هذا الأمر لتعيين إحداثيات التركيب الحالي .

الشكل العام لهذا الأمر:

```
void glTexCoord1{sifd}{v} (TYPE s);
void glTexCoord2{sifd}{v} (TYPE s, TYPE t);
void glTexCoord3{sifd}{v} (TYPE s, TYPE t, TYPE r);
void glTexCoord4{sifd}{v} (TYPE s, TYPE t, TYPE r, TYPE q);
```

3. glTexGen

يستخدم هذا الأمر للتحكم بعملية توليد التركيب الحالي .

الشكل العام لهذا الأمر:

```
void glTexGen{ifd}{v} (GLenum coord, GLenum pname, TYPE param);
```

يتحكم بكيفية توليد إحداثيات التركيب .

coord: يعين إحداثيات تركيب معين .قيمه : *GL_S, GL_T, GL_R, GL_Q*.

pname: يعين تسمية لتابع توليد إحداثيات التركيب . يجب أن تكون قيمته *GL_TEXTURE_GEN_MODE* . يمكن استخدام القيم التالية في حالة استخدام طريقة الشعاع *v* : *GL_TEXTURE_GEN_MODE, GL_OBJECT_PLANE, GL_EYE_PLANE* .

param: يعين وسيط توليد تركيب أحادي القيمة .قيمه *GL_OBJECT_LINEAR, GL_EYE_LINEAR, GL_SPHERE_MAP* .

4. glTexImage

يستخدم هذا الأمر لتحديد صورة تركيب أحادية البعد أو ثنائية البعد .

الشكل العام لهذا الأمر:

*void glTexImage1D (GLenum target, GLint level, GLint components, GLsizei width, GLint border, GLenum format, GLenum type, const GLvoid *pixels);*
*void glTexImage2D (GLenum target, GLint level, GLint components, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *pixels);*

5. glGetTexEnv, glGetTexGen, glGetTexImage, glGetTexLevelParameter, glGetTexParameter

تستخدم هذه الأوامر للحصول على قيم الوسائط المتعلقة بالتراكيب .

الشكل العام لهذه الأوامر:

*void glGetTexEnv{if}v (GLenum target, GLenum pname, TYPE *params);*

يعيد وسائط بيئة التركيب .

target: يعين بيئة تركيب . قيمته يجب أن تكون *GL_TEXTURE_ENV* .

pname: يعين تسمية لوسيط بيئة تركيب. قيمته يجب ان تكون

GL_TEXTURE_ENV_MODE أو *GL_TEXTURE_ENV_COLOR*.

param: يعيد البيانات المطلوبة .

*void glGetTexGen{ifd}v (GLenum coord, GLenum pname, TYPE *params);*

يعيد وسائط توليد إحداثيات التركيب .

coord: يعين إحداثيات تركيب معين .قيمه : *GL_S, GL_T, GL_R, GL_Q*.

pname: يعين تسمية للقيم المراد إعادتها . يجب أن تكون قيمته

GL_TEXTURE_GEN_MODE أو اسم أحد معدلات مستوي توليد التركيب التالية :

GL_OBJECT_PLANE or GL_EYE_PLANE.

param: يعيد البيانات المطلوبة .

*void glGetTexImage (GLenum target, GLint level, GLenum format, GLenum type, GLvoid *pixels);*

يعيد صورة تركيب معين .

target: يحدد التركيب المراد الحصول عليه .القيم المقبولة : *GL_TEXTURE_1D*

GL_TEXTURE_2D, .

level: يحدد مستوى رقم التفاصيل للصورة المطلوبة . يمثل المستوى *0* مستوى الصورة الرئيسي .

format: يحدد شكل النقاط الضوئية للبيانات المعادة . الأشكال الممكنة :

GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_RGB, GL_RGBA, GL_LUMINANCE, GL_LUMINANCE_ALPHA.

type:يحدد نوع نقاط ضوئية للبيانات المعادة . قيمه

GL_UNSIGNED_BYTE, GL_BYTE, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT, GL_FLOAT.

pixels: يعيد صورة التركيب . يجب أن يكون مؤشراً إلى مصفوفة نوعها حسب *type* .

*void glGetTexLevelParameter{if}v (GLenum target, GLint level, GLenum pname, TYPE *params);*

يعيد قيم وسائط تركيب من أجل مستوى تفصيل معين .

target: يحدد التركيب المراد الحصول عليه .القيم المقبولة : *GL_TEXTURE_1D*

GL_TEXTURE_2D, .

level: يحدد مستوى رقم التفاصيل للصورة المطلوبة . يمثل المستوى *0* مستوى الصورة الرئيسي .

pname: يحدد اسم وسيط التركيب .قيمه الممكنة :

GL_TEXTURE_WIDTH, GL_TEXTURE_HEIGHT, GL_TEXTURE_COMPONENTS, GL_TEXTURE_BORDER

param: يعيد البيانات المطلوبة .

*void glGetTexParameter{if}v (GLenum target, GLenum pname, TYPE *params);*

يعيد قيم وسائط تركيب معين .

target: يحدد التركيب المراد الحصول عليه .القيم المقبولة : *GL_TEXTURE_1D*
,GL_TEXTURE_2D .

pname: يحدد اسم وسيط التركيب . قيمه المحتملة :

GL_TEXTURE_MAG_FILTER, GL_TEXTURE_MIN_FILTER, GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T, GL_TEXTURE_BORDER_COLOR.

params: يعيد وسائط التركيب .

مجموعة الضباب Fog

1. glFog

يستخدم هذا الأمر لتعيين وسائط الضباب .

الشكل العام لهذا الأمر:

void glFog{if}{v} (GLenum pname, TYPE param);

مجموعة عمليات الذاكرة المؤقتة للإطار Frame Buffer Operations

1. glScissor, glAlphaFunc, glStencilFunc, glStencilOp, glDepthFunc

تستخدم هذه الأوامر للتحكم بفحص مرحلة *fragment* .

الشكل العام لهذه الأوامر:

void glScissor (GLint x, GLint y, GLsizei width, GLsizei height);

يعرف مربع قص .

x, y: يحددان الزاوية اليسارية السفلية لمربع القص . قيمة التهيئة *(0,0)* .

width, height: يحددان عرض وارتفاع صندوق القص .

void (GLenum func, GLclampf ref);

يحدد تابع فحص الشفافية (ألفا) .

func: يحدد تابع مقارنة الشفافية . قيمه المحتملة (القيمة الافتراضية
(GL_ALWAYS :

GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_GREATER, GL_NOTEQUAL, GL_GEQUAL, GL_ALWAYS .

ref: يعين قيمة مرجعية تقارن معها قيمة ألفا القادمة . تتوضع هذه القيمة ضمن المجال *[0,1]* . تمثل القيمة *0* أقل قيمة محتملة لألفا والقيمة *1* أعلى قيمة محتملة لألفا . القيمة الافتراضية *0* .

void glStencilFunc (GLenum func, GLint ref, GLuint mask);

يعين تابع وقيمة مرجعية للفحص الحاجب .

func: يحدد تابع الفحص . قيمه المحتملة :

GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_GREATER, GL_NOTEQUAL, GL_GEQUAL, GL_ALWAYS .

ref: يعين قيمة مرجعية من اجل الفحص الحاجب. تتوضع هذه القيمة ضمن المجال *[0,2n - 1]* . تمثل *n* عدد مستويات البتات في الذاكرة المؤقتة الحاجبة .

mask: يحدد قناع يطبق عليه المعامل المنطقي *AND* مع كل من القيمة المرجعية والقيمة الحاجبة المخزنة عند انتهاء عملية الفحص .

void glStencilOp (GLenum fail, GLenum pass, GLenum zpass);

يعين أفعال فحص الحجب.

fail: يعين الفعل الواجب اتخاذه عند فشل فحص الحجب .القيم الممكنة :

GL_KEEP, GL_ZERO, GL_REPLACE, GL_INCR, GL_DECR, GL_INVERT.

zfail: يعين الفعل الواجب اتخاذه عند نجاح فحص الحجب ولكن مع فشل فحص العمق. القيم الممكنة:

GL_KEEP, GL_ZERO, GL_REPLACE, GL_INCR, GL_DECR, and GL_INVERT.

zpass: يعين الفعل الواجب اتخاذه عند نجاح فحص الحجب و فحص العمق. القيم الممكنة :

GL_KEEP, GL_ZERO, GL_REPLACE, GL_INCR, GL_DECR, GL_INVERT.

void glDepthFunc (GLenum func);

يعين القيمة المستخدمة لمقارنة الذاكرة المؤقتة للعمق .

func: يحدد تابع مقارنة العمق . القيم المحتملة (القيمة الافتراضية *GL_LESS*):

GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_GREATER, GL_NOTEQUAL, GL_GEQUAL, GL_ALWAYS.

2. glBlendFunc, glLogicOp

يستخدم هذان الأمران لمزج قيم مرحلة *fragment* ومرحلة *frame buffer* .

الشكل العام لهذين الأمرين:

```
void glBlendFunc (GLenum sfactor, GLenum dfactor);
void glLogicOp (GLenum opcode);
```

يحدد عملية نقاط ضوئية منطقية من أجل تصيير فهرس اللون .

opcode: يعين ثابتاً لتحديد العملية المنطقية . القيم الممكنة :

GL_CLEAR, GL_SET, GL_COPY, GL_COPY_INVERTED, GL_NOOP, GL_INVERT, GL_AND, GL_NAND, GL_OR, GL_NOR, GL_XOR, GL_EQUIV, GL_AND_REVERSE, GL_AND_INVERTED, GL_OR_REVERSE, GL_OR_INVERTED.

3. glClear

يستخدم هذا الأمر لمسح ذاكرة مؤقتة واحدة أو أكثر .

الشكل العام لهذا الأمر:

```
void glClear (GLbitfield mask);
```

4. glClearAccum, glClearColor, glClearDepth glClearIndex, glClearStencil

تستخدم هذه الأوامر لتعيين قيم مسح اللون والعمق والحجب .

الشكل العام لهذه الأوامر:

```
void glClearAccum (GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha);
void glClearColor (GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);
void glClearDepth (GLclampd depth);
void glClearIndex (GLfloat c);
void glClearStencil (GLint s);
```

5. glDrawBuffer, glIndexMask, glColorMask, glDepthMask, glStencilMask

تستخدم هذه الأوامر لتفعيل الذواكر المؤقتة من أجل عملية الكتابة .

الشكل العام لهذه الأوامر:

void glDrawBuffer (GLenum mode);

يحدد نوع الذاكرة المؤقتة التي سيرسم إليها .

mode:يحدد حتى أربع ذواكر لونية مؤقتة للرسم إليها .. القيم الممكنة :

GL_NONE, GL_FRONT_LEFT, GL_FRONT_RIGHT, GL_BACK_LEFT, GL_BACK_RIGHT, GL_FRONT, GL_BACK, GL_LEFT, GL_RIGHT, GL_FRONT_AND_BACK, GL_AUXi.

قيمة *i* بين *0* و *GL_AUX_BUFFERS -1* .

void glIndexMask (GLuint mask);

يتحكم بعملية كتابة خانات مستقلة ضمن الذواكر المؤقتة لفهرس اللون .

mask: يعين قناع خانة لتفعيل أو تعطيل كتابة خانات مستقلة ضمن الذواكر المؤقتة لفهرس اللون .

void glColorMask (GLboolean red, GLboolean green, GLboolean blue, GLboolean alpha);

يفعل أو يعطل كتابة المكونات اللونية إلى ذاكرة الإطار .

red, green, blue, alpha: يحدد فيما إذا كان يمكن أو لا يمكن كتابة اللون الأحمر والأخضر والأزرق والشفافية إلى الذاكرة المؤقتة للإطار . القيمة الافتراضية لجميع المكونات *GL_TRUE* للدلالة على إمكانية الكتابة إلى الذاكرة المؤقتة للإطار.

void glDepthMask (GLboolean flag);

يفعل أو يعطل الكتابة إلى الذاكرة المؤقتة للعمق .

flag: يحدد فيما إذا كانت الذاكرة المؤقتة للعمق مفعلة . إذا كانت *flag=0* عند ذلك ستعطل الكتابة على الذاكرة المؤقتة للعمق وستفعل الكتابة من اجل القيم الأخرى وهي الوضعية الابتدائية .

void glStencilMask (GLuint mask);

يتحكم بكتابة خانات مستقلة إلى مستويات الحجب .

mask: يحدد قناع خانة لتفعيل أو تعطيل كتابة خانات مستقلة ضمن مستويات الحجب .

6. glAccum

يستخدم هذا الأمر للعمل على الذاكرة المؤقتة التراكمية .

الشكل العام لهذا الأمر:

void glAccum (GLenum op, GLfloat value);

يعمل على الذاكرة المؤقتة التراكمية .

op: يحدد عملية الذاكرة المؤقتة التراكمية.القيم الممكنة :

GL_ACCUM, GL_LOAD, GL_ADD, GL_MULT, GL_RETURN.

value: يحدد قيمة فاصلة عائمة مستخدمة في عملية الذاكرة المؤقتة التراكمية .

مجموعة المقيمات *Evaluators*

1. glMap1, glMap2

يستخدم هذان الأمران لتعريف مقيمات ببعد واحد أو بعدين .

الشكل العام لهذين الأمرين:

*void glMap1{fd} (GLenum target, TYPE u1, TYPE u2, GLint stride, GLint order, const TYPE *points);*

يعرف مقيماً أحادي البعد .

target: يحدد نوع القيم المولدة بواسطة المقيم . القيم الممكنة:

GL_MAP1_VERTEX_3, GL_MAP1_VERTEX_4, GL_MAP1_INDEX, GL_MAP1_COLOR_4, GL_MAP1_NORMAL, GL_MAP1_TEXTURE_COORD_1, GL_MAP1_TEXTURE_COORD_2, GL_MAP1_TEXTURE_COORD_3, GL_MAP1_TEXTURE_COORD_4.

u1, u2: يحدد تنظيماً خطياً لـــ *u* .

stride: يحدد عدد الخانات العائمة أو المزدوجة بين بداية نقطة تحكم وبداية نقطة تحكم اخرى ضمن بنية البيانات المشار إليها بـــ *points* .

order: يحدد عدد نقاط التحكم . يجب أن تكون قيمته موجبة .

points: يحدد مؤشراً إلى مصفوفة نقاط التحكم .

*void glMap2{fd} (GLenum target, TYPE u1, TYPE u2, GLint ustride, GLint uorder, TYPE v1, TYPE v2, GLint vstride,GLint vorder, const TYPE *points);*

يعرف مقيماً ثنائي البعد .

target: يحدد نوع القيم المولدة بواسطة المقيم . القيم الممكنة:

GL_MAP1_VERTEX_3, GL_MAP1_VERTEX_4, GL_MAP1_INDEX, GL_MAP1_COLOR_4, GL_MAP1_NORMAL, GL_MAP1_TEXTURE_COORD_1, GL_MAP1_TEXTURE_COORD_2, GL_MAP1_TEXTURE_COORD_3, GL_MAP1_TEXTURE_COORD_4.

u1, u2: يحدد تنظيماً خطياً لـــ *u* .

ustride: يحدد عدد الخانات العائمة أو المزدوجة بين بداية نقطة تحكم *Rij* وبداية نقطة تحكم أخرى *R (i+1) j* ، حيث أن *i,j* مؤشرات إلى نقاط التحكم *u,v* .

uorder: يحدد أبعاد مصفوفة نقاط التحكم وفق المحور *u* . يجب أن تكون القيمة موجبة .

v1, v2: يحدد تنظيماً خطياً لـــ *v* .

vstride: يحدد عدد الخانات العائمة أو المزدوجة بين بداية نقطة تحكم *Rij* وبداية نقطة تحكم أخرى *R (i+1) j* ، حيث أن *i,j* مؤشرات إلى نقاط التحكم *u,v* .

vorder: يحدد أبعاد مصفوفة نقاط التحكم وفق المحور *v* . يجب أن تكون القيمة موجبة .

points: يحدد مؤشراً إلى مصفوفة نقاط التحكم .

2. glMapGrid1, glMapGrid2, glEvalMesh1, glEvalMesh2, glEvalPoint1, glEvalPoint2

تستخدم هذه الأوامر لتوليد وتقييم سلسلة من القيم للخرائط .

الشكل العام لهذه الأوامر:

void glMapGrid1{fd} (GLint un, TYPE u1, TYPE u2);

يعرف عنصراً هندسياً متشابكاً *mesh* أحادي البعد .

un : يحدد عدد الأجزاء ضمن مجال الشبكة *[u1, u2]* . قيمته موجبة .

u1, u2: يحدد تنظيماً من أجل قيم مجال الشبكة الصحيحة المتوضعة بين *i=0* و *i=un*

void glMapGrid2{fd} (GLint un, TYPE u1, TYPE u2, GLint vn, TYPE v1, TYPE v2);

يعرف عنصراً هندسياً متشابكاً *mesh* ثنائي البعد .

un : يحدد عدد الأجزاء ضمن مجال الشبكة *[u1, u2]* . قيمته موجبة .

u1, u2: يحدد تنظيماً من أجل قيم مجال الشبكة الصحيحة المتوضعة بين *i=0* و *i=un* .

vn : يحدد عدد الأجزاء ضمن مجال الشبكة *[v1, v2]* . قيمته موجبة .

v1, v2: يحدد تنظيماً من أجل قيم مجال الشبكة الصحيحة المتوضعة بين *i=0* و *i=vn* .

void glEvalMesh1 (GLenum mode, GLint i1, GLint i2);

يحسب أبعاد الشبكة الأحادية للنقاط أو الخطوط .

mode: يحدد هل سيحسب أبعاد الشبكة للنقاط أم للخطوط . قيمه *GL_POINT , GL_LINE*.

i1, i2: يحدد أول وآخر مجال قيم صحيحة لمتحول مجال الشبكة *i* .

void glEvalMesh2(GLenum mode,Glint i1,GLint i2,GLint j1, GLint j2);

يحسب أبعاد الشبكة الثنائية للنقاط أو الخطوط .

mode: يحدد هل سيحسب أبعاد الشبكة للنقاط أم للخطوط أم المضلعات . قيمه *GL_POINT , GL_LINE, GL_FILL*.

i1, i2: يحدد أول وآخر مجال قيم صحيحة لمتحول مجال الشبكة *i* .

j1,j2: يحدد أول وآخر مجال قيم صحيحة لمتحول مجال الشبكة *j* .

void glEvalPoint1 (GLint i);

يولد ويقيم نقطة وحيدة في عنصر شبكي *mesh* .

i: يحدد قيمة صحيحة لمتحول مجال الشبكة *i* .

void glEvalPoint2 (GLint i, GLint j);

يولد ويقيم نقطة وحيدة في عنصر شبكي *mesh* .

i: يحدد قيمة صحيحة لمتحول مجال الشبكة *i* .

j: يحدد قيمة صحيحة لمتحول مجال الشبكة *j* .

3. glEvalCoord1, glEvalCoord2

يستخدم هذان الأمران لتقييم خرائط أحادية البعد أو ثنائية البعد عند مجال إحداثيات محدد .

الشكل العام لهذين الأمرين:

void glEvalCoord1{fd}{v} (TYPE u);

يقيم تفعيل خرائط أحادية البعد .

u: يحدد قيمة تكون مجال إحداثيات *u* .تشكل هذه القيمة التابع الأساسي لأوامر *glMap1 , glMap2* السابقة .

void glEvalCoord2{fd}{v} (TYPE u, TYPE v);

يقيم تفعيل خرائط ثنائية البعد .

u: يحدد قيمة تكون مجال إحداثيات *u* .تشكل هذه القيمة التابع الأساسي لأوامر *glMap1 , glMap2* السابقة .

v: يحدد قيمة تكون مجال إحداثيات *v* .تشكل هذه القيمة التابع الأساسي لأمر *glMap2* السابق .

4. glGetMap

يستخدم هذا الأمر للحصول على قيم وسائط المقيمات .

الشكل العام لهذا الأمر:

*void glGetMap{idf}v (GLenum target, GLenum query, TYPE *v);*

يعيد وسائط المقيم .

target: يحدد اسم خريطة .القيم المحتملة :

GL_MAP1_COLOR_4, GL_MAP1_INDEX, GL_MAP1_NORMAL, GL_MAP1_TEXTURE_COORD_1, GL_MAP1_TEXTURE_COORD_2, GL_MAP1_TEXTURE_COORD_3, GL_MAP1_TEXTURE_COORD_4, GL_MAP1_VERTEX_3, GL_MAP1_VERTEX_4, GL_MAP2_COLOR_4 , GL_MAP2_INDEX, GL_MAP2_NORMAL, GL_MAP2_TEXTURE_COORD_1, GL_MAP2_TEXTURE_COORD_2, GL_MAP2_TEXTURE_COORD_3, GL_MAP2_TEXTURE_COORD_4, GL_MAP2_VERTEX_3, GL_MAP2_VERTEX_4.

query: يحدد الوسيط المراد إعادته . القيم المحتملة :

GL_COEFF, GL_ORDER, and GL_DOMAIN

v: يعيد البيانات المطلوبة .

مجموعة التحديد والتغذية العكسية *Selection and Feedback*

1. glRenderMode, glSelectBuffer, glFeedbackBuffer

تستخدم هذه الأوامر لتحديد نمط التصيير والذاكرة المؤقتة الموافقة .

الشكل العام لهذه الأوامر:

GLint glRenderMode (GLenum mode);

يحدد نمط مسح *rasterization* .

mode: يحدد نمط مسح . القيم الممكنة: *GL_RENDER* (القيمة الافتراضية) و *GL_SELECT* و *GL_FEEDBACK* .

*void glSelectBuffer (GLsizei size, GLuint *buffer);*

يبدأ ذاكرة مؤقتة جديدة من اجل قيم نمط التحديد .

size: يحدد حجم الذاكرة المؤقتة .

buffer: يعيد البيانات المحددة .

*void glFeedbackBuffer (GLsizei size, GLenum type, GLfloat *buffer);*

يتحكم بنمط التغذية العكسية .

size: يحدد أعظم عدد من القيم التي يمكن كتابتها في الذاكرة المؤقتة .

type: يعين ثابتاً يصف المعلومات التي ستعاد من أجل كل رأس . القيم الممكنة :

GL_2D, GL_3D, GL_3D_COLOR, GL_3D_COLOR_TEXTURE,GL_4D_COLOR_TEXTURE.

buffer: يعيد بيانات التغذية العكسية .

2. glPassThrough

يستخدم هذا الأمر لتزويد علامة *token* لنمط التغذية العكسية .

الشكل العام لهذا الأمر:

void glPassThrough (GLfloat token);

يضع مسجل في الذاكرة المؤقتة للتغذية العكسية .

token: يحدد قيمة المسجل التي ستوضع في الذاكرة المؤقتة للتغذية العكسية .

3. glInitNames, glLoadName, glPushName, glPopName

تستخدم هذه الأوامر للتحكم بمكدس الاسم الخاص بعملية للتحديد.

الشكل العام لهذه الأوامر:

void glInitNames (void);

يهيئ مكدس الاسم (يستخدم مكدس الاسم خلال نمط التحديد للسماح لمجموعة أوامر تصيير بأن تكون معرفة بشكل فريد).

void glLoadName (GLuint name);

يحمّل اسماً إلى مكدس الاسم .

name: الاسم الذي سيحل مكان القيمة الموجودة في قمة مكدس الاسم .

void glPushName (GLuint name);

يدفع اسماً إلى مكدس الاسم .

name: الاسم المراد دفعه إلى مكدس الاسم .

void glPopName (void);

يسحب اسماً من قمة مكدس الاسم .

مجموعة لوائح الإظهار *Display Lists*

1. glNewList, glEndList, glDeleteLists

تستخدم هذه الأوامر لإنشاء لوائح الإظهار وحذفها .

الشكل العام لهذه الأوامر:

void glNewList (GLuint list, GLenum mode);
void glEndList (void);
void glDeleteLists (GLuint list, GLsizei range);

2. glCallList, glCallLists

يستخدم هذان الأمران لتنفيذ لائحة إظهار واحدة أو مجموعة من لوائح الإظهار .

الشكل العام لهذين الأمرين:

void glCallList (GLuint list);
*void glCallLists (GLsizei n, GLenum type, const GLvoid *lists);*

ينفذ عدة لوائح إظهار .

n: يحدد عدد لوائح الإظهار المراد تنفيذها .

type: يحدد نوع القيم في اللوائح .القيم المحتملة :

GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT, GL_2_BYTES, GL_3_BYTES, and GL_4_BYTES.

lists: يحدد مؤشراً إلى مصفوفة من إزاحات الأسماء ضمن لائحة الإظهار .

3. glGenLists, glIsList, glListBase

تستخدم هذه الأوامر لإدارة فهرسة لوائح الإظهار .

الشكل العام لهذه الأوامر:

GLuint glGenLists (GLsizei range);
GLboolean glIsList (GLuint list);

يفحص وجود لائحة إظهار .

list: يحدد اسم لائحة الإظهار المراد التأكد من وجودها .

void glListBase (GLuint base);

يعين أساس لائحة الإظهار من أجل الأمر *glCallLists* .

base: يحدد قيمة إزاحة صحيحة تضاف إلى إزاحات *glCallLists* لتوليد أسماء لوائح إظهار . قيمة التهيئة *0* .

مجموعة الأنماط والتنفيذ Modes and Execution

1. glEnable, glDisable, glIsEnabled

تستخدم هذه الأوامر لتفعيل الأنماط وإلغاء تفعيلها والاستعلام عن نمط معين .

الشكل العام لهذه الأوامر:

```
void glEnable (GLenum cap);
void glDisable (GLenum cap);
GLboolean glIsEnabled (GLenum cap);
```

يفحص فيما إذا تم تفعيل الإمكانية أم لا .

cap: يشير إلى ثابت يحدد إمكانية *OpenGL* .

2. glFinish

يستخدم هذا الأمر لانتظار إنهاء تنفيذ جميع أوامر *OpenGL* بشكل كامل .

الشكل العام لهذا الأمر:

```
void glFinish (void);
```

3. glFlush

يستخدم هذا الأمر لإجبار تنفيذ جميع أوامر *OpenGL* الصادرة .

الشكل العام لهذا الأمر:

```
void glFlush (void);
```

يجبر أوامر *OpenGL* على التنفيذ خلال فترة زمنية محددة .

4. glHint

يستخدم هذا الأمر لتطبيق بعض التحكمات على جودة وسرعة عمل الصور في *OpenGL* .

الشكل العام لهذا الأمر:

```
void glHint (GLenum target, GLenum mode);
```

مجموعة استعلامات الحالة State Queries

1. glGetError, glGetString

يستخدم هذان الأمران للحصول على معلومات عن خطأ ما أو لتحديد اتصال *OpenGL* الحالي .

الشكل العام لهذين الأمرين:

GLenum glGetError (void);

يعيد معلومات عن الأخطاء.

*const GLubyte * glGetString (GLenum name);*

يعيد سلسلة نصية تصف اتصال *OpenGL* الحالي .

name: يحدد ثابت . القيم الممكنة :

GL_VENDOR, GL_RENDERER, GL_VERSION, GL_EXTENSIONS.

2. glGetBooleanv, glGetDoublev, glGetFloatv, glGetIntegerv

تستخدم هذه الأوامر للاستعلام عن متحولات الحالة.

الشكل العام لهذه الأوامر:

*void glGetBooleanv (GLenum pname, GLboolean *params);*

يعيد قيمة منطقية لوسيط محدد . القيمة المعادة إما *GL_FALSE* أو *GL_TRUE*

pname: يحدد قيمة الوسيط المراد إعادها . وله الكثير من القيم المحتملة .

params: يعيد قيمة الوسيط المحدد .

*void glGetDoublev (GLenum pname, GLdouble *params);*

يعيد قيمة فاصلة عائمة لوسيط محدد . القيمة المعادة إما *GL_FALSE* أو *GL_TRUE* من أجل القيم المنطقية أما الأعداد الصحيحة فتحول إلى فاصلة عائمة .

*void glGetFloatv (GLenum pname, GLfloat *params);*

يعيد قيمة فاصلة عائمة لوسيط محدد . القيمة المعادة إما *GL_FALSE* أو *GL_TRUE* من أجل القيم المنطقية أما الأعداد الصحيحة فتحول إلى فاصلة عائمة .

*void glGetIntegerv (GLenum pname, GLint *params);*

يعيد قيمة صحيحة لوسيط محدد . القيمة المعادة إما *GL_FALSE* أو *GL_TRUE* من أجل القيم المنطقية ، أما أعداد الفاصلة العائمة فتدور إلى أقرب رقم صحيح .

3. glPushAttrib, glPopAttrib

يستخدم هذان الأمران لحفظ واسترجاع مجموعة من متحولات الحالة .

الشكل العام لهذين الأمرين:

void glPushAttrib (GLbitfield mask);

يأخذ هذا التابع معاملاً واحداً اسمه *mask* يشير إلى أي مجموعة من متحولات الحالة ستخزن في مكدس المواصفات .

mask: يحدد قناعاً يشير إلى المواصفات الواجب تخزينها . القيم المحتملة :

GL_ACCUM_BUFFER_BIT , GL_COLOR_BUFFER_BIT ,GL_CURRENT_BIT , GL_DEPTH_BUFFER_BIT , GL_ENABLE_BIT , GL_EVAL_BIT , GL_FOG_BIT, GL_HINT_BIT, GL_LIGHTING_BIT, GL_LINE_BIT, GL_LIST_BIT, GL_PIXEL_MODE_BIT, GL_POINT_BIT, GL_POLYGON_BIT, GL_POLYGON_STIPPLE_BIT, GL_SCISSOR_BIT, GL_STENCIL_BUFFER_BIT, GL_TEXTURE_BIT, GL_TRANSFORM_BIT, GL_VIEWPORT_BIT.

```
void glPopAttrib (void);
```

يستعيد قيم متحولات الحالة المخزنة وفق آخر أمر *glPushAttrib* .

الملحق ب

المكتبة GLU

تؤمن ***OpenGL*** مجموعة قوية لكن قليلة من العمليات ، ويجب على جميع عمليات الرسم على مستوى أعلى الاعتماد على هذه الأوامر. تعمل المكتبة ***GLU*** على تسهيل العمل البرمجي الرسومي على مبرمج الرسوميات ، فهي تتضمن العديد من الإجراءات التي تبنى اعتماداً على أوامر ***OpenGL*** . نقدم فيما يلي شرحاً لهذه الأوامر وسنعمل على تقسيمها إلى مجموعات حسب وظيفتها :

مجموعة صور التراكيب Texture Images

1. gluScaleImage

يعمل هذا الأمر على توسيع أو تقليص حجم الصورة .

الشكل العام لهذا الأمر:

int gluScaleImage (GLenum format*, GLint* widthin*, GLint* heightin*,GLenum* typein*, const void* *datain*, GLint* widthout*, GLint* heightout*, GLenum* typeout*, void* *dataout*);*

format : يحدد شكل بيانات النقاط الضوئية.القيم الممكنة لهذا الوسيط :

GL_COLOR_INDEX, GL_STENCIL_INDEX, GL_DEPTH_COMPONENT, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_RGB, GL_RGBA, GL_LUMINANCE, GL_LUMINANCE_ALPHA.

widthin, heightin: يحدد هذان الوسيطان عرض وارتفاع الصورة المصدر المراد تغيير حجمها .

typein: يحدد نوع البيانات من اجل الوسيط *datain* . قيمه:

GL_UNSIGNED_BYTE, GL_BYTE, GL_BITMAP, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT, GL_FLOAT.

datain : يحدد مؤشراً إلى الصورة المصدر .

widthout, heightout: يحدد هذان الوسيطان عرض وارتفاع الصورة الهدف .

typeout: يحدد نوع البيانات من اجل الوسيط *dataout*. قيمه:

GL_UNSIGNED_BYTE, GL_BYTE, GL_BITMAP, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT, GL_FLOAT.

dataout : يحدد مؤشراً إلى الصورة الهدف .

2. gluBuild1DMipmaps, gluBuild2DMipmaps

يستخدم هذان الأمران لتوليد تركيب من صورة بغض النظر عن أبعاد هذه الصورة.

الشكل العام لهذين الأمرين:

int gluBuild1DMipmaps (GLenum target*, GLint* components*, GLint* width*, GLenum* format*, GLenum* type*, void* *data*);*
int gluBuild2DMipmaps (GLenum target*, GLint* components*, GLint* width*, GLint* height*, GLenum* format*, GLenum* type*, void* *data*);*

target: يحدد التركيب الهدف . قيمته *GL_TEXTURE_1D* من أجل الأمر *gluBuild1DMipmaps* و *GL_TEXTURE_2D* من أجل الأمر *gluBuild2DMipmaps* .

components: يحدد عدد المكونات اللونية في التركيب . قيمه *1,2,3,4* .

width: يحدد عرض صورة التركيب .

format: يحدد شكل بيانات النقاط الضوئية . قيمه:

GL_COLOR_INDEX, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_RGB, GL_RGBA, GL_LUMINANCE,GL_LUMINANCE_ALPHA

type: يحدد نوع البيانات من أجل بيانات التركيب *data* . قيمه :

GL_UNSIGNED_BYTE, GL_BYTE, GL_BITMAP, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT,GL_FLOAT.

data : يحدد مؤشراً إلى بيانات الصورة في الذاكرة .

مجموعة تحويل الإحداثيات *Coordinate Transformation*

1. gluOrtho2D, gluPerspective, gluPickMatrix gluLookAt

تستخدم هذه الأوامر لإنشاء مصفوفات تحويل الإسقاط *Projection* وتحويل العرض *Viewing* .

الشكل العام لهذه الأوامر:

void gluOrtho2D (GLdouble left, GLdouble right, GLdouble bottom,GLdouble top);

يحدد مصفوفة تحويل إسقاط متعامد ثنائي البعد.

left, right : يحدد إحداثيات مستويات القطع الشاقولية اليسارية واليمينية .

bottom, top : يحدد إحداثيات مستويات القطع الأفقية العلوية والسفلية .

void gluPerspective (GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar);

يعين مصفوفة التحويل المنظوري .

fovy : يحدد زاوية حقل الرؤية بالدرجات في اتجاه المحور *y* .

aspect : يحدد شكل النسبة المستخدمة لتحديد حقل الرؤية في الاتجاه *x* .شكل النسبة عبارة عن نسبة *x* (العرض) إلى *y* (الارتفاع).

zNear:يحدد المسافة من عين الناظر إلى مستوي القطع القريب(القيمة موجبة دائماً).

zFar: يحدد المسافة من عين الناظر إلى مستوي القطع البعيد (القيمة موجبة دائماً).

void gluPickMatrix (GLdouble x, GLdouble y, GLdouble width, GLdouble height, GLint viewport[4]);

يعرف منطقة التقاط .

x, y: يحددان مركز منطقة الالتقاط ضمن إحداثيات الإطار .

width, height: يحددان عرض وارتفاع منطقة الالتقاط ضمن إحداثيات الإطار .

viewport: يحدد المسقط *viewport* الحالي .

void gluLookAt (GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz);

يعرف هذا التابع تحويل *viewing* .

eyex, eyey, eyez: تحدد هذه الوسائط موقع عين الناظر .

centerx, centery, centerz: تحدد هذه الوسائط موقع النقطة المرجعية المراد النظر إليها .

upx, upy, upz : تحدد هذه الوسائط اتجاه الشعاع العلوي .

2. gluProject, gluUnProject

يستخدم هذان الأمران لتحويل إحداثيات العناصر إلى إحداثيات الشاشة وبالعكس .

الشكل العام لهذين الأمرين:

*int gluProject (GLdouble objx, GLdouble objy, GLdouble objz, const GLdouble modelMatrix[16], const GLdouble projMatrix[16], const GLint viewport[4], GLdouble *winx, GLdouble *winy, GLdouble *winz);*

يحول إحداثيات العنصر إلى إحداثيات الشاشة .

objx, objy, objz: تحدد هذه الوسائط إحداثيات العنصر .

modelMatrix : يحدد مصفوفة *modelMatrix* الحالية .

projMatrix: يحدد مصفوفة الإسقاط *projection* الحالية .

viewport: يحدد *viewport* الحالي .

winx, winy, winz: تعيد هذه الوسائط إحداثيات الإطار المحسوبة .

*int gluUnProject (GLdouble winx, GLdouble winy, GLdouble winz, const GLdouble modelMatrix[16], const GLdouble projMatrix[16], const GLint viewport[4], GLdouble *objx, GLdouble *objy, GLdouble *objz);*

يحول إحداثيات الشاشة إلى إحداثيات العنصر .

winx, winy, winz: تحدد هذه الوسائط إحداثيات الإطار .

modelMatrix : يحدد مصفوفة *modelMatrix* الحالية .

projMatrix: يحدد مصفوفة الإسقاط *projection* الحالية .

viewport: يحدد *viewport* الحالي .

objx, objy, objz: تعيد هذه الوسائط إحداثيات العنصر المحسوبة .

مجموعة تعبئة ترصيع ورسم المضلعات *Polygon Tessellation*

1. gluNewTess, gluTessCallback, gluDeleteTess

تستخدم هذه الأوامر لإدارة العناصر المرصعة.

الشكل العام لهذه الأوامر:

*GLUtriangulatorObj*gluNewTess (void);*

ينشئ عنصراً مرصعاً .

*void gluTessCallback(GLUtriangulatorObj*tobj,GLenum which, void (*fn)());*

يعرف استدعاءً خلفياً من اجل عنصر مرصع .

tobj : يحدد العنصر المرصع (المنشئ بواسطة الأمر *gluNewTess*).

which : يحدد الاستدعاء الخلفي المعرف . قيمه :

GLU_BEGIN, GLU_EDGE_FLAG, GLU_VERTEX, GLU_END, and GLU_ERROR.

fn : يحدد التابع المراد استدعاءه .

*void gluDeleteTess (GLUtriangulatorObj *tobj);*

يدمر العنصر المرصع .

tobj : يحدد العنصر المرصع المراد تدميره (المنشئ بواسطة الأمر *gluNewTess*).

2. gluBeginPolygon, gluEndPolygon, gluNextContour, gluTessVertex

تستخدم هذه الأوامر لوصف المضلع المدخل.

الشكل العام لهذه الأوامر:

*void gluBeginPolygon (GLUtriangulatorObj *tobj);*

يحدد بداية رسم المضلع.

tobj : يحدد العنصر المرصع (المنشئ بواسطة الأمر *gluNewTess*).

*void gluEndPolygon (GLUtriangulatorObj *tobj);*

يحدد نهاية رسم المضلع.

tobj : يحدد العنصر المرصع (المنشئ بواسطة الأمر *gluNewTess*)

*void gluNextContour (GLUtriangulatorObj *tobj, GLenum type);*

يحدد بداية محيط عنصر جديد.

tobj : يحدد العنصر المرصع (المنشئ بواسطة الأمر *gluNewTess*).

type :يحدد نوع المحيط المعرف .قيمه

GLU_EXTERIOR, GLU_INTERIOR, GLU_UNKNOWN, GLU_CCW, and GLU_CW.

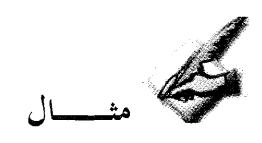
*void gluTessVertex (GLUtriangulatorObj *tobj, GLdouble v[3], void *data);*

يحدد نقطة على المضلع.

tobj : يحدد العنصر المرصع (المنشئ بواسطة الأمر *gluNewTess*).

v: يحدد موقع النقطة.

data : يحدد مؤشراً يمرر عائداً إلى المستخدم من خلال الاستدعاء الخلفي للنقطة .



مثـــــال

يرسم هذا المثال عنصراً هندسياً رباعي الأضلاع وضمنه ثقب مثلثي الشكل:

```
gluBeginPolygon(tobj);
  gluTessVertex(tobj, v1, v1);
  gluTessVertex(tobj, v2, v2);
  gluTessVertex(tobj, v3, v3);
  gluTessVertex(tobj, v4, v4);
  gluNextContour(tobj, GLU_INTERIOR);
  gluTessVertex(tobj, v5, v5);
  gluTessVertex(tobj, v6, v6);
  gluTessVertex(tobj, v7, v7);
gluEndPolygon(tobj);
```

مجموعة العناصر الهندسية *Quadric Objects*

1. gluNewQuadric, gluDeleteQuadric gluQuadricCallback

تستخدم هذه الأوامر لإدارة العناصر *Quadric*.

الشكل العام لهذه الأوامر:

```
GLUquadricObj* gluNewQuadric (void);
```

ينشئ عنصراً *Quadric*.

```
void gluDeleteQuadric (GLUquadricObj *state);
```

يحذف عنصراً *Quadric*.

state: يحدد عنصر *Quadric* المراد حذفه (العنصر المنشئ بواسطة الأمر *gluNewQuadric*).

```
void gluQuadricCallback (GLUquadricObj *qobj, GLenum which, void (*fn)());
```

يعرف استدعاءً خلفياً من أجل عنصر *Quadric* ما .

tobj : يحدد عنصر *Quadric* المراد تعريف استدعاء خلفي له (العنصر المنشئ بواسطة الأمر *gluNewQuadric*).

which : يحدد الاستدعاء الخلفي المعرف . قيمته الوحيدة *GLU_ERROR* .

fn : يحدد التابع المراد استدعاءه .

2. gluQuadricNormals, gluQuadricTexture, gluQuadricOrientation, gluQuadricDrawStyle

تستخدم هذه الأوامر للتحكم بتصيير العناصر *Quadric*.

الشكل العام لهذه الأوامر:

*void gluQuadricNormals (GLUquadricObj *quadObject, GLenum normals);*

يحدد أنواع الأشعة الاعتيادية المخصصة *Quadric*.

quadObject: يحدد العنصر*Quadric*(المنشئ بواسطة الأمر *gluNewQuadric*).

normals: يحدد نوع الشعاع الاعتيادي المرغوب . قيمه :

GLU_NONE, GLU_FLAT, GLU_SMOOTH

*void gluQuadricTexture (GLUquadricObj *quadObject, GLboolean textureCoords);*

يحدد فيما إذا كان الإكساء بالتراكيب مطلوباً أم لا من أجل *Quadric*.

quadObject: يحدد العنصر *Quadric*(المنشئ بواسطة الأمر*gluNewQuadric*).

textureCoords: يحدد علماً للإشارة إلى توليد إحداثيات التركيب أم لا .

*void gluQuadricOrientation (GLUquadricObj *quadObject, GLenum orientation);*

يحدد الاتجاه الداخلي والخارجي *Quadric*.

quadObject: يحدد العنصر*Quadric*(المنشئ بواسطة الأمر *gluNewQuadric*).

orientation: يحدد الاتجاه المطلوب . قيمه *GLU_OUTSIDE and* *GLU_INSIDE* .

*void gluQuadricDrawStyle (GLUquadricObj *quadObject, GLenum drawStyle);*

يحدد نمط الرسم المرغوب من أجل رسم *Quadric*.

quadObject: يحدد العنصر الرباعي (المنشئ بواسطة الأمر *gluNewQuadric*).

drawStyle: يحدد نمط الرسم المرغوب . قيمه :

GLU_FILL, GLU_LINE, GLU_SILHOUETTE, and GLU_POINT

3. gluCylinder, gluDisk, gluPartialDisk, gluSphere

تستخدم هذه الأوامر لرسم عناصر هندسية مختلفة اعتماداً على *Quadric*.

الشكل العام لهذه الأوامر:

*void gluCylinder (GLUquadricObj *qobj, GLdouble baseRadius,GLdouble topRadius, GLdouble height, GLint slices, GLint stacks);*

يرسم اسطوانة .

qobj: يحدد العنصر *Quadric* (المنشئ بواسطة الأمر *gluNewQuadric*).

baseRadius: يحدد نصف قطر الاسطوانة عند قيمة *z=0* .

topRadius: يحدد نصف قطر الاسطوانة عند قيمة *z=height* .

height: ارتفاع الاسطوانة .

slices: يحدد عدد التقسيمات حول المحور *z* .

stacks: يحدد عدد التقسيمات على طول المحور *z*

*void gluDisk (GLUquadricObj *qobj, GLdouble innerRadius, GLdouble outerRadius, GLint slices, GLint loops);*

يرسم قرصاً .

qobj: يحدد العنصر *Quadric* (المنشئ بواسطة الأمر *gluNewQuadric*).

innerRadius: نصف القطر الداخلي للقرص .

outerRadius: نصف القطر الخارجي للقرص .

slices: يحدد عدد التقسيمات حول المحور *z* .

loops: يحدد عدد الحلقات المتداخلة (التقسيمات العرضية).

*void gluPartialDisk (GLUquadricObj *qobj, GLdouble innerRadius,GLdouble outerRadius, GLint slices, GLint loops, GLdouble startAngle, GLdouble sweepAngle);*

يرسم قوساً من قرص .

qobj: يحدد العنصر *Quadric* (المنشئ بواسطة الأمر *gluNewQuadric*).

innerRadius: نصف القطر الداخلي للقرص .

outerRadius: نصف القطر الخارجي للقرص .

slices: يحدد عدد التقسيمات حول المحور *z* .

loops: يحدد عدد الحلقات المتداخلة (التقسيمات العرضية).

startAngle: يحدد زاوية البداية بالدرجات لجزء القرص المراد رسمه .

sweepAngle: يحدد زاوية الإزالة بالدرجات لجزء القرص المراد رسمه

```
void gluSphere (GLUquadricObj *qobj, GLdouble radius, GLint slices,GLint stacks);
```

يرسم كرة.

qobj: يحدد العنصر *Quadric* (المنشئ بواسطة الأمر *gluNewQuadric*).

radius: يحدد نصف قطر الكرة .

slices: يحدد عدد التقسيمات حول المحور *z* (تشبه خطوط الطول).

stacks: يحدد عدد التقسيمات على طول المحور *z* (تشبه دوائر العرض) .

مجموعة منحنيات وسطوح NURBS (NURBS Curves and Surfaces)

1. gluNewNurbsRenderer, gluDeleteNurbsRenderer, gluNurbsCallback

تستخدم هذه الأوامر لإدارة عناصر *NURBS*.

الشكل العام لهذه الأوامر:

```
GLUnurbsObj* gluNewNurbsRenderer (void);
```

ينشئ عنصر *NURBS* .

```
void gluDeleteNurbsRenderer (GLUnurbsObj *nobj);
```

يحذف عنصر *NURBS* .

nobj: يحدد عنصر *NURBS* المراد حذفه .

```
void gluNurbsCallback (GLUnurbsObj *nobj, GLenum which, void (*fn)());
```

يعرف استدعاءً خلفياً لعنصر *NURBS* .

which : يحدد الاستدعاء الخلفي المعرف . قيمته الوحيدة *GLU_ERROR* .

fn : يحدد التابع المراد استدعاءه .

2. gluBeginCurve, gluEndCurve, gluNurbsCurve

تستخدم هذه الأوامر لإنشاء منحني *NURBS*.

الشكل العام لهذه الأوامر:

*void gluBeginCurve (GLUnurbsObj *nobj);*

يحدد بداية منحني *NURBS* .

nobj: يحدد عنصر *NURBS* (المنشئ بواسطة الأمر *gluNewNurbsRenderer*)

*void gluEndCurve (GLUnurbsObj *nobj);*

يحدد نهاية منحني *NURBS* .

nobj: يحدد عنصر *NURBS* (المنشئ بواسطة الأمر *gluNewNurbsRenderer*)

*void gluNurbsCurve (GLUnurbsObj *nobj, GLint nknots, GLfloat *knot, GLint stride, GLfloat *ctlarray, GLint order, GLenum type);*

يحدد شكل منحني *NURBS* .

nobj: يحدد عنصر *NURBS* (المنشئ بواسطة الأمر *gluNewNurbsRenderer*)

nknots:يحدد عدد العقد في الربطة الواحدة *knot*.

knot: يحدد مصفوفة العقد.

stride: يحدد مقدار الإزاحة بين نقطتي تحكم في المنحني .

ctlarray:يحدد مؤشراً إلى مصفوفة تحوي نقاط التحكم .يجب أن تتوافق إحداثيات النقاط مع النوع *type* لهذا الأمر.

order: يحدد ترتيب منحني *NURBS* .

type: يحدد نوع المنحني .إذا كان المنحني معرفاً بين *gluBeginCurve* و*gluEndCurve* فستكون قيمة *type* *GL_MAP1_VERTEX_3* أو *GL_MAP1_COLOR_4*، أما إذا كان المنحني معرفاً بين *gluBeginTrim* و

gluEndTrim فستكون قيمة *type* *GLU_MAP1_TRIM_2* أو *GLU_MAP1_TRIM_3* .

3. gluBeginSurface, gluNurbsSurface

يستخدم هذان الأمران لإنشاء سطح *NURBS*.

الشكل العام لهذين الأمرين:

```
void gluBeginSurface (GLUnurbsObj *nobj);void gluEndSurface
(GLUnurbsObj *nobj);
```

يحدد بداية سطح *NURBS* .

nobj: يحدد عنصر *NURBS* (المنشئ بواسطة الأمر *gluNewNurbsRenderer*)

```
void gluNurbsSurface (GLUnurbsObj *nobj, GLint uknot_count,
GLfloat *uknot, GLint vknot_count, GLfloat *vknot, GLint u_stride,
GLint v_stride, GLfloat *ctlarray, GLint sorder, GLint torder, GLenum
type);
```

يحدد شكل سطح *NURBS* .

nobj: يحدد عنصر *NURBS* (المنشئ بواسطة الأمر *gluNewNurbsRenderer*)

sknot_count: يحدد عدد العقد في الاتجاه *u* .

sknot:يحدد مصفوفة من *sknot_count* .

tknot_count: يحدد عدد العقد في الاتجاه *v* .

tknot: يحدد مصفوفة من *tknot_count* .

s_stride: يحدد إزاحة بين نقاط التحكم في الاتجاه *u* .

t_stride: يحدد إزاحة بين نقاط التحكم في الاتجاه *v* .

ctlarray: يحدد مصفوفة تحوي نقاط التحكم من أجل سطح *NURBS* .

sorder: يحدد ترتيب سطح *NURBS* وفق الاتجاه *u*.

torder: يحدد ترتيب سطح *NURBS* وفق الاتجاه *v*.

type: يحدد نوع سطح *NURBS* . يمكن أن تكون قيمة *type* أي نوع مقيمات صحيحة ثنائية البعد مثل *GL_MAP2_VERTEX_3* أو *GL_MAP2_COLOR_4* .

4. gluBeginTrim, gluEndTrim, gluPwlCurve

تستخدم هذه الأوامر لتحديد منطقة قطع.

الشكل العام لهذه الأوامر:

*void gluBeginTrim (GLUnurbsObj *nobj);*

يحدد بداية منطقة قطع *NURBS* .

nobj: يحدد عنصر *NURBS* (المنشئ بواسطة الأمر *gluNewNurbsRenderer*)

*void gluEndTrim (GLUnurbsObj *nobj);*

يحدد نهاية منطقة قطع *NURBS* .

nobj: يحدد عنصر *NURBS* (المنشئ بواسطة الأمر *gluNewNurbsRenderer*)

*void gluPwlCurve (GLUnurbsObj *nobj, GLint count, GLfloat *array, GLint stride, GLenum type);*

يحدد منحني قطع عينة من سطح *NURBS* .

nobj: يحدد عنصر *NURBS* (المنشئ بواسطة الأمر *gluNewNurbsRenderer*)

count: يحدد عدد النقاط على المنحني .

array: يحدد مصفوفة إحداثيات نقاط المنحني .

stride:يحدد إزاحة بين النقاط على المنحني .

type: يحدد نوع المنحني . قيمت إما *GLU_MAP1_TRIM_2* أو *GLU_MAP1_TRIM_3*.

5. gluLoadSamplingMatrices, gluNurbsProperty, gluGetNurbsProperty

تستخدم هذه الأوامر للتحكم بتصيير عناصر *NURBS*.

الشكل العام لهذه الأوامر:

*void gluLoadSamplingMatrices (GLUnurbsObj *nobj, const GLfloat modelMatrix[16], const GLfloat projMatrix[16], const GLint viewport[4]);*

يحدد عينات *NURBS* والمصفوفات المنتقاة .

nobj: يحدد عنصر *NURBS* (المنشئ بواسطة الأمر *gluNewNurbsRenderer*)

modelMatrix: يحدد مصفوفة *modelMatrix* .

projMatrix: يحدد مصفوفة تحويل الإسقاط *projection* .

viewport: يحدد *viewport* .

*void gluNurbsProperty (GLUnurbsObj *nobj, GLenum property, GLfloat value);*

يعين خصائص *NURBS* .

nobj: يحدد عنصر *NURBS* (المنشئ بواسطة الأمر *gluNewNurbsRenderer*)

property: يحدد الخاصية المراد إسنادها.القيم المحتملة لهذا الوسيط :

GLU_SAMPLING_TOLERANCE, GLU_DISPLAY_MODE, GLU_CULLING, GLU_AUTO_LOAD_MATRIX.

value: يحدد القيمة المسندة للخاصية السابقة المحددة .

*void gluGetNurbsProperty (GLUnurbsObj *nobj, GLenum property, GLfloat *value);*

الحصول على خصائص *NURBS* .

nobj: يحدد عنصر *NURBS* (المنشئ بواسطة الأمر *gluNewNurbsRenderer*)

property: يحدد الخاصية التي تم الوصول إلى قيمتها . القيم المحتملة :

GLU_CULLING, GLU_SAMPLING_TOLERANCE, GLU_DISPLAY_MODE, GLU_AUTO_LOAD_MATRIX.

value: يحدد مؤشراً إلى الموقع الذي كتبت فيه القيمة من أجل الخاصية السابقة .

مجموعة معالجة الأخطاء *Error Handling*

هناك تابع وحيد لمعالجة الأخطاء يعمل على انتاج سلسة نصية للخطأ اعتماداً على النص البرمجي لأخطاء *OpenGL* .

الشكل العام لهذا الأمر:

const GLubyte gluErrorString (GLenum errorCode);*

يولد سلسة خطأ نصية اعتماداً على النص البرمجي لأخطاء مكتبة *OpneGL* أو مكتبة *GLU* .

errorCode: يحدد النص البرمجي لأخطاء مكتبة *OpneGL* أو مكتبة *GLU* .

الملحق ج

المكتبة GLUT

تعمل هذه المكتبة على إنجاز الكثير من العمليات ضمن *OpenGL*، و قد استفدنا منها بشكل كبير ضمن هذا الكتاب. سنشرح الآن أهم أوامر هذه المكتبة و سنضع هذه الأوامر ضمن مجموعات حسب وظيفتها:

مجموعة التهيئة Initialization

1. glutInit

يستخدم هذا التابع (الأمر) لتهيئة مكتبة *GLUT* و فتح جلسة مع نظام التشغيل *Windows* .

الشكل العام لهذا الأمر:

```
void glutInit(int *argcp,char **argv);
```

الوسيط `argcp` عبارة عن مؤشر إلى متحول البرنامج `argc` غير المعدل المتوضع ضمن التابع *main* . تُحدث القيمة المشار إليها بواسطة `argcp` لدى العودة، لأن التابع `glutInit` يستخرج أي سطر أوامر مفهوم من قبل المكتبة *GLUT* .

الوسيط `argv` عبارة عن متحول برنامج غير معدل متوضع ضمن *main*. تُحدث هنا أيضاً بيانات هذا المتحول لأن*GLUT* تستخرج أي سطر أوامر مفهوم من قبل المكتبة *GLUT*.

2. glutInitWindowPosition,glutInitWindowSize

يستخدم هذان الأمران لتعيين موقع و حجم الإطار.

الشكل العام لهذين الأمرين:

```
void glutInitWindowSize(int width,int height);
void glutInitWindowPosition(int x,int y);
```

width : عرض الإطار بالنقطة الضوئية *Pixel*.

height : ارتفاع الإطار بالنقطة الضوئية.

x : قيمة الإحداثي *x* لموقع الإطار بالنقطة الضوئية.

y : قيمة الإحداثي *y* لموقع الإطار بالنقطة الضوئية.

3. glutInitDisplayMode

يستخدم هذا التابع لتهيئة نمط الإظهار.

الشكل العام للتابع:

```
void glutInitDisplayMode(unsigned int mode);
```

mode: يحدد نمط الإظهار، و يستخدم معه المعامل *OR*. يمكن أن تكون قيم mode (نمط الإظهار) إحدى القيم التالية:

القيمة	الشرح
GLUT_RGBA	يحدد إطاراً بنمط *RGBA* وهو الخيار الافتراضي.
GLUT_RGB	حالة خاصة من نمط GLUT_RGBA .
GLUT_INDEX	يحدد إطاراً بنمط فهرس اللون.
GLUT_SINGLE	يحدد إطاراً بذاكرة مؤقتة وحيدة وهذا الخيار افتراضي.
GLUT_DOUBLE	يحدد إطاراً بذاكرة مؤقتة مزدوجة.
GLUT_ACCUM	يحدد إطاراً بذاكرة مؤقتة تراكمية.
GLUT_ALPHA	يحدد إطاراً بمكون ألفا (الشفافية) للذاكرة المؤقتة للون.
GLUT_DEPTH	يحدد إطاراً بذاكرة عمق مؤقتة.
GLUT_STENCIL	يحدد إطاراً بذاكرة حاجبة مؤقتة.
GLUT_MULTISAMPLE	يحدد إطاراً يدعم نماذج متعددة.
GLUT_STEREO	يحدد إطار ستريو.
GLUT_LUMINANCE	يحدد إطاراً بنمط لون يحدد إضاءة ذاتية.

مجموعة معالجة الأحداث

1. glutMainLoop

يدخل هذا الأمر إلى حلقة معالجة الأحداث لمكتبة *GLUT*. يستدعى هذا الأمر مرة واحدة في البرنامج. يتم تكرار الإجراء الذي يستدعي هذا الأمر بشكل متكرر و بلا نهاية.

الشكل العام لهذا الأمر:

void glutMainLoop(void);

مجموعة إدارة الإطار

تدعم *OpenGL* نوعين من الإطارات: الإطارات ذات المستوى الأعلى و الإطارات الفرعية.

1. glutCreateWindow

ينشئ إطاراً ذو مستوى أعلى.

الشكل العام للأمر:

```
int glutCreateWindow(char *name);
```

name : عبارة عن سلسلة رموز نصية تستخدم لكتابة اسم الإطار.

2. glutCreateSubWindow

ينسئ هذا الأمر إطاراً فرعيا.

الشكل العام للأمر:

```
int glutCreateSubWindow(int win,int x,int y,int width,int height);
```

win : يحدد رقم تعريف الأب للإطار الفرعي هذا.

x : يحدد الإحداثي x بالنقطة الضوئية للإطار الفرعي بالنسبة إلى موقع الإطار الأب .

y : يحدد الإحداثي y بالنقطة الضوئية للإطار الفرعي بالنسبة إلى موقع الإطار الأب.

width : يحدد عرض الإطار بالنقطة الضوئية.

height : يحدد ارتفاع الإطار بالنقطة الضوئية.

3. glutSetWindow,glutGetWindow

يعين الأمر *glutSetWindow* الإطار الحالي، في حين يعين الأمر*glutGetWindow* رقم تعريف خاص بالإطار الحالي.

الشكل العام لهذين الأمرين:

```
void glutSetWindow(int win);
int glutGetWindow(void);
```

win : يحدد رقم تعريف إطار لصنع الإطار الحالي.

4. glutDestroyWindow

يدمر (يحذف) الإطار المحدد.

الشكل العام للأمر:

```
void glutDestroyWindow(int win);
```

win : رقم تعريف الإطار المراد تدميره.

5. glutPostRedisplay

يعمل هذا التابع على إعادة إظهار الإطار الحالي حسب الحاجة، و يستفاد منه في حالة تحريك العناصر، إذ لا تظهر حركة العناصر دون استدعاء هذا التابع.

الشكل العام لهذا التابع:

```
void glutPostRedisplay(void);
```

6. glutSwapBuffers

يعمل هذا الأمر على تبديل الذواكر المؤقتة للإطار الحالي إذا كانت من نوع الذاكرة المؤقتة المزدوجة، يستدعى هذا الأمر بشكل دائم في نهاية تابع الرسم.

الشكل العام لهذا الأمر:

```
void glutSwapBuffers(void);
```

7. glutPositionWindow

يستخدم هذا الأمر لتغيير موقع الإطار الحالي.

الشكل العام لهذا الأمر:

```
void glutPositionWindow(int x,int y);
```

x : الإحداثي x الجديد للإطار بالنقطة الضوئية.

y : الإحداثي y الجديد للإطار بالنقطة الضوئية.

8. glutReshapeWindow

يستخدم هذا الأمر لتغيير حجم الإطار الحالي.

الشكل العام لهذا الأمر:

```
void glutReshapeWindow(int width,int height);
```

width : العرض الجديد للإطار بالنقطة الضوئية.

height : الارتفاع الجديد للإطار بالنقطة الضوئية.

9. glutFullScreen

يستخدم هذا الأمر لجعل الإطار الحالي يملئ الشاشة.

الشكل العام لهذا الأمر:

```
void glutFullScreen(void);
```

10. glutPopWindow,glutPushWindow

يغير هذان الأمران ترتيب التكديس للإطار الحالي مع أقربائه. و يعمل هذان الأمران على الإطارات عالية المستوى و الإطارات الفرعية.

الشكل العام لهذين الأمرين:

```
void glutPopWindow(void);
void glutPushWindow(void);
```

11. glutShowWindow,glutHideWindow,glutIconifyWindow

تستخدم هذه الأوامر لتغيير حالة الإطار الحالي. حيث يستخدم التابع الأول لإظهار الإطار، أما التابع الثاني فيخفي الإطار، في حين يستخدم التابع الثالث لوضع رمز للإطارات عالية المستوى، و منع وضع رموز للإطارات الفرعية.

الشكل العام لهذه الأوامر:

```
void glutShowWindow(void);
void glutHideWindow(void);
void glutIconifyWindow(void);
```

12. glutSetWindowTitle,glutSetIconTitle

يستخدم هذان الأمران لتغيير عنوان الإطار و الرمز بشكل متتالي للإطار الحالي عالي المستوى.

الشكل العام لهذين الأمرين:

```
void glutSetWindowTitle(char *name);
void glutSetIconTitle(char *name);
```

name: عبارة عن سلسلة نصية من رموز الآسكي تستخدم لتعيين اسم إطار أو اسم رمز للإطار الحالي.

13. glutSetCursor

يغير هذا الأمر صورة مؤشر الفأرة للإطار الحالي.

الشكل العام لهذا الأمر:

```
void glutSetCursor(int cursor);
```

cursor : اسم صورة المؤشر المراد التغيير إليه. يبين الجدول التالي أسماء صور المؤشرات مع الصورة الموافقة للمؤشر.

اسم صورة المؤشر	الصورة
GLUT_CURSOR_RIGHT_ARROW	
GLUT_CURSOR_LEFT_ARROW	
GLUT_CURSOR_INFO	
GLUT_CURSOR_DESTROY	
GLUT_CURSOR_HELP	?
GLUT_CURSOR_CYCLE	
GLUT_CURSOR_SPRAY	
GLUT_CURSOR_WAIT	
GLUT_CURSOR_TEXT	I
GLUT_CURSOR_CROSSHAIR	+
GLUT_CURSOR_UP_DOWN	↕
GLUT_CURSOR_LEFT_RIGHT	↔
GLUT_CURSOR_TOP_SIDE	↑
GLUT_CURSOR_BOTTOM_SIDE	↕
GLUT_CURSOR_LEFT_SIDE	↔
GLUT_CURSOR_RIGHT_SIDE	↔
GLUT_CURSOR_TOP_LEFT_CORNER	⤡
GLUT_CURSOR_TOP_RIGHT_CORNER	⤢
GLUT_CURSOR_BOTTOM_RIGHT_CORNER	⤡
GLUT_CURSOR_BOTTOM_LEFT_CORNER	⤢
GLUT_CURSOR_FULL_CROSSHAIR	+
GLUT_CURSOR_NONE	لا يوجد مؤشر
GLUT_CURSOR_INHERIT	استخدام مؤشر الإطار الأب

مجموعة إدارة القوائم

تدعم *GLUT* قوائم منسدلة متزايدة. تستخدم هذه القوائم لجعل المستخدم ينتقي عدة أنماط خلال البرنامج.

1. glutCreateMenu

ينشئ قائمة منسدلة جديدة.

الشكل العام لهذا الأمر:

```
int glutCreateMenu(void (*func)(int value));
```

func : تابع الاستدعاء الخلفي *callback* للقائمة التي استدعيت عندما تم اختيار عنصر (مدخل) من القائمة.

value : تمرر إلى الاستدعاء الخلفي و تحدد بواسطة القيمة المعينة من أجل عنصر القائمة المحدد.

2. glutSetMenu,glutGetMenu

يستخدم الأمر *glutSetMenu* لتعيين القائمة الحالية، في حين يعين الأمر *glutGetMenu* رقم تعريف القائمة الحالية.

الشكل العام لهذين الأمرين:

```
void glutSetMenu(int menu);
int glutGetMenu(void);
```

menu : رقم تعريف يعين لصنع القائمة الحالية.

3. glutDestroyMenu

يدمر القائمة المحددة. و لا يؤثر هذا الأمر على القوائم الفرعية من هذه القائمة.

الشكل العام لهذا الأمر:

```
void glutDestroyMenu(int menu);
```

menu : رقم تعريف القائمة المراد تدميرها.

4. glutAddMenuEntry

يضيف هذا الأمر عنصراً جديداً إلى أسفل القائمة الحالية.

الشكل العام لهذا الأمر:

```
void glutAddMenuEntry(char *name,int value);
```

name : سلسلة نصية تمثل اسماً يظهر على عنصر القائمة

value : قيمة تعاد إلى تابع الاستدعاء الخلفي للقائمة إذا تم تحديد هذا العنصر.

5. glutAddSubMenu

يضيف هذا الأمر قادح قائمة فرعية إلى أسفل القائمة الحالية.

الشكل العام لهذا الأمر:

```
void glutAddSubMenu(char *name,int menu);
```

name : سلسلة رموز نصية بالآسكي تظهر ضمن القائمة و يتم من خلالها فتح القائمة الفرعية.

menu : رقم تعريف للقائمة الفرعية يمكن من تحديد عناصرها.

6. glutRemoveMenuItem

يحذف هذا الأمر عنصر القائمة المحدد.

الشكل العام لهذا الأمر:

```
void glutRemoveMenuItem(int entry);
```

entry : فهرس يشير إلى عناصر القائمة الحالية (الرقم *1* لعنصر القائمة الأعلى).

7. glutAttachMenu,glutDetachMenu

يلحق الأمر*glutAttachMenu* زر فأرة إلى معرف القائمة الحالية للإطار الحالي.

يفصل الأمر *glutDetachMenu* زر فأرة ملحق من الإطار الحالي.

الشكل العام لهذين الأمرين:

```
void glutAttachMenu(int button);
void glutDetachMenu(int button);
```

button : الزر المراد إلحاقه إلى قائمة أو فصله عنها. عند ضغط هذا الزر تفتح القائمة المنسدلة. قيمه هي:

GLUT_LEFT_BUTTON,GLUT_MIDDLE_BUTTON,
GLUT_RIGHT_BUTTON.

مجموعة تسجيل الاستدعاء الخلفي

تدعم المكتبة *GLUT* عدداً من الاستدعاءات الخلفية من أجل الاستجابة للأحداث. هناك ثلاثة أنواع من الاستدعاءات الخلفية: الإطار *window* و القائمة *menu* و عام *global* . تستخدم استدعاءات الإطار عند إعادة رقم الإطار عند تغير شكل عرضه أو عند إتاحة إدخال إليه . استدعاء الإطار يعين بواسطة الأمر *glutCreateMenu* .

1. glutDisplayFunc

يعين الاستدعاء الخلفي للإظهار الخاص بالإطار الحالي.

الشكل العام للأمر:

```
void glutDisplayFunc(void (*func)(void));
```

func : تابع الاستدعاء الخلفي للإظهار .

2. glutReshapeFunc

يعين الاستدعاء الخلفي لإعادة رسم الأشكال للإطار الحالي.

الشكل العام للتابع:

```
void glutReshapeFunc(void (*func)(int width,int height));
```

func : تابع الاستدعاء الخلفي لإعادة رسم الأشكال.

width : عرض الإطار الجديد بالنقطة الضوئية.

height : ارتفاع الإطار الجديد بالنقطة الضوئية.

3. glutKeyboardFunc

يعين الاستدعاءات الخلفية للوحة المفاتيح من أجل الإطار الحالي. يستفاد من هذا التابع في إضافة تحكمات للإطار بواسطة أزرار من لوحة المفاتيح.

الشكل العام للتابع:

```
void glutKeyboardFunc(void (*func)(unsigned char key,int x,int y));
```

func : تابع الاستدعاء الخلفي الجديد للوحة المفاتيح.

Key: رمز الآسكي المولد عند ضغط مفتاح من لوحة المفاتيح .

X,y : تشير إلى موقع مؤشر الفأرة ضمن الإطار بالإحداثيات النسبية وذلك عند ضغط المفتاح المحدد .

4. glutMouseFunc

يعين استدعاء الفأرة من أجل الإطار الحالي. يستفاد من هذا التابع في إنجاز عمليات على الإطار عند ضغط أو تحرير أحد أزرار الفأرة.

الشكل العام لهذا الأمر:

```
void glutMouseFunc(void (*func)(int button,int state,int x,int y));
```

func : تابع استدعاء خلفي جديد للفأرة.

button : يشير إلى الزر المفعل و قيمه:

GLUT_LEFT_BUTTON,GLUT_MIDDLE_BUTTON,
GLUT_RIGHT_BUTTON.

state : يحدد الاستدعاء عند ضغط أو تحرير زر الفأرة و قيمه:

GLUT_UP , GLUT_DOWN.

X,y : تشير إلى الإحداثيات النسبية للإطار عند تغير حالة زر الفأرة.

5. glutSpecialFunc

يعين استدعاءات لوحة مفاتيح خاصة من أجل الإطار الحالي.

الشكل العام لهذا الأمر:

```
void glutSpecialFunc(void (*func)(int key,int x,int y));
```

func : تابع استدعاء لوحة المفاتيح الخاص الجديد.

key :عبارة عن ثابت يستدعي مفتاح خاص من لوحة المفاتيح.

X,y : تشير إلى الإحداثيات النسبية للفأرة في الإطار عند ضغط المفتاح الخاص.

قيم key مبينة في الجدول التالي:

قيمة الثابت	المفتاح المقابل من لوحة المفاتيح
GLUT_KEY_F1	المفتاح الوظيفي F1
GLUT_KEY_F2	المفتاح الوظيفي F2
GLUT_KEY_F3	المفتاح الوظيفي F3
GLUT_KEY_F4	المفتاح الوظيفي F4
GLUT_KEY_F5	المفتاح الوظيفي F5
GLUT_KEY_F6	المفتاح الوظيفي F6
GLUT_KEY_F7	المفتاح الوظيفي F7
GLUT_KEY_F8	المفتاح الوظيفي F8
GLUT_KEY_F9	المفتاح الوظيفي F9
GLUT_KEY_F10	المفتاح الوظيفي F10
GLUT_KEY_F11	المفتاح الوظيفي F11
GLUT_KEY_F12	المفتاح الوظيفي F12
GLUT_KEY_LEFT	مفتاح السهم (الحركة) اليساري
GLUT_KEY_UP	مفتاح السهم (الحركة) العلوي
GLUT_KEY_RIGHT	مفتاح السهم (الحركة) اليميني
GLUT_KEY_DOWN	مفتاح السهم (الحركة) السفلي
GLUT_KEY_PAGE_UP	مفتاح انتقال الصفحات لأعلى PageUp
GLUT_KEY_PAGE_DOWN	مفتاح انتقال الصفحات لأسفل PageDown
GLUT_KEY_HOME	مفتاح الانتقال للبداية Home
GLUT_KEY_END	مفتاح الانتقال للنهاية End
GLUT_KEY_INSERT	مفتاح الحشر Insert

6. glutTimerFunc

يسجل هذا الأمر استدعاء لمؤقت زمني يقدح بعد فترة زمنية خاصة مقدرة بالميلي ثانية.

الشكل العام لهذا الأمر:

*void glutTimerFunc(unsigned int msecs,void (*func)(int value),value);*

msecs : الفترة الزمنية المنقضية بالميلي ثانية قبل استدعاء المؤقت الزمني.

func : وظيفة استدعاء المؤقت الزمني.

value : رقم صحيح يمرر إلى استدعاء المؤقت الزمني.

مجموعة إدارة فهرس اللون

تدعم *OpenGL* نمطي الألوان *RGBA* و فهرس اللون. يفضل استخدام نمط *RGBA* لإمكانياته الكبيرة.

1. glutSetColor

يعيد هذا الأمر لون مدخل خريطة اللون ضمن الإطار الحالي.

الشكل العام لهذا الأمر:

void glutSetColor(int cell,GLfloat red,GLfloat green,GLfloat blue);

cell : رقم فهرس خلية اللون (يبدأ بالرقم صفر).

red : كثافة اللون الأحمر (المجال بين *0.0* و *1.0*).

green : كثافة اللون الأخضر (المجال بين *0.0* و *1.0*).

blue : كثافة اللون الأزرق (المجال بين *0.0* و *1.0*).

2. glutGetColor

يسترجع هذا الأمر مكونات اللون الأحمر و الأخضر و الأزرق من مدخل فهرس لون ضمن خريطة لون معطاة من أجل الإطار الحالي.

الشكل العام لهذا الأمر:

GLfloat glutGetColor(int cell,int component);

`cell` : رقم فهرس خلية اللون (يبدأ بالرقم صفر).

component : إحدى القيم *GLUT_RED* أو *GLUT_GREEN* أو *GLUT_BLUE* .

3. glutCopyColormap

ينسخ خريطة اللون المنطقية من إطار خاص إلى الإطار الحالي.

الشكل العام لهذا الأمر:

```
void glutCopyColormap(int win);
```

win : رقم تعريف الإطار المراد نسخ خريطة اللون المنطقية منه.

مجموعة تصيير الخطوط

تدعم *OpenGL* نوعين من تصيير الخطوط: خطوط *Stroke* التي يتم فيها تصيير كل رمز على شكل مجموعة من الخطوط المقسمة *lines*. و خطوط الصور النقطية *Bitmap* التي يكون فيها كل رمز عبارة عن صورة نقطية.

1. glutBitmapCharacter

يصير هذا الأمر رمز صورة نقطية باستخدام *OpenGL* .

الشكل العام لهذا الأمر:

```
void glutBitmapCharacter(void *font,int character);
```

font : خط الصورة النقطية المراد استخدامه.

character :الرمز المراد تصييره (لا يقتصر على *8* خانات).

الخطوط المتاحة بواسطة هذا الأمر مبينة في الجدول التالي:

الخط	الشرح
GLUT_BITMAP_8_BY_13	خط بعرض ثابت يتوضع ضمن مستطيل أبعاده 8x13pixel .
GLUT_BITMAP_9_BY_15	خط بعرض ثابت يتوضع ضمن مستطيل أبعاده 9x15pixel.
GLUT_BITMAP_TIMES_ROMAN_10	خط Times Roman بحجم 10 نقطة.
GLUT_BITMAP_TIMES_ROMAN_24	خط Times Roman بحجم 24 نقطة.
GLUT_BITMAP_HELVETICA_10	خط Helvetica بحجم 10 نقطة .
GLUT_BITMAP_HELVETICA_12	خط Helvetica بحجم 12 نقطة .
GLUT_BITMAP_HELVETICA_18	خط Helvetica بحجم 18 نقطة .

2. glutBitmapWidth

يعيد هذا الأمر عرض رمز صورة نقطية.

الشكل العام لهذا الأمر:

```
int glutBitmapWidth(GLUTbitmapFont font,int character);
```

font : خط الصورة النقطية المستخدم.

character : الرمز المراد إعادة عرضه (لا يقتصر على *8* خانات).

3. glutStrokeCharacter

يصير رمز*Stroke* باستخدام *OpenGL* .

الشكل العام لهذا الأمر:

```
void glutStrokeCharacter(void *font,int character);
```

font : الخط المراد استخدامه.

character : الرمز المراد تصييره (لا يقتصر على *8* خانات).

الخطوط المتاحة مبينة في الجدول التالي:

الخط	الشرح
GLUT_STROKE_ROMAN	خط Roman Simplex بأبعاد تناسبية لرموز الآسكي من 32 وحتى 127 .أعلى ارتفاع للرمز هنا 119.05 وحدة ، وأقل انحدار للرمز 33.33 وحدة .
GLUT_STROKE_MONO_ROMAN	خط Roman Simplex بأبعاد وحيدة لرموز الآسكي من 32 وحتى 127 .أعلى ارتفاع للرمز هنا 119.05 وحدة ، وأقل انحدار للرمز 33.33 وحدة .عرض كل رمز 104.76 وحدة .

4. glutStrokeWidth

يعيد هذا الأمر عرض رمز*Stroke*.

الشكل العام لهذا الأمر:

```
int glutStrokeWidth(GLUTstrokeFont font,int character);
```

font : خط *Stroke* المستخدم.

character : الرمز المراد إعادة عرضه (لا يقتصر على *8* خانات).

رسم العناصر الهندسية

تتضمن المكتبة *GLUT* أوامر جاهزة لبناء عناصر ثلاثية البعد. هذه العناصر تشبيه عناصر المكتبة *AUX* التي استخدمناها ضمن فصول الكتاب لرسم العناصر الهندسية ثلاثية البعد. يمكنك تجريب استخدام العناصر التي سنشرحها بعد قليل بدلاً من عناصر المكتبة *AUX*

1. glutSolidSphere,glutWireSphere

يرسم هذان الأمران كرة مصمتة و كرة سلكية على التوالي.

الشكل العام لهذين الأمرين:

```
void glutSolidSphere(GLdouble radius,GLint slices,GLint stacks);
void glutWireSphere(GLdouble radius,GLint slices,GLint stacks);
```

radius : نصف قطر الكرة

slices : عدد التقسيمات حول المحور *Z* (تشبه خطوط الطول).

stacks : عدد التقسيمات على طول المحور *Z* (تشبه دوائر العرض).

2. glutSolidCube,glutWireCube

يرسم هذان الأمران مكعب مصمت و سلكي على التوالي.

الشكل العام لهذين الأمرين:

```
void glutSolidCube(GLdouble size);
void glutWireCube(GLdouble size);
```

size : طول ضلع المكعب.

3. glutSolidCone,glutWireCone

يرسم هذان الأمران مخروط مصمت و سلكي على التوالي.

الشكل العام لهذين الأمرين:

```
void glutSolidCone(GLdouble base,GLdouble height,
GLint slices,GLint stacks);
void glutWireCone(GLdouble base,GLdouble height,
GLint slices,GLint stacks);
```

base : نصف قطر قاعدة المخروط.

height : ارتفاع المخروط.

slices : عدد التقسيمات حول المحور *Z* (تشبه خطوط الطول).

stacks : عدد التقسيمات على طول المحور *Z* (تشبه دوائر العرض الطول).

4. glutSolidTorus,glutWireTorus

يرسم هذان الأمران طارة مصمتة و سلكية على التوالي.

الشكل العام لهذين الأمرين:

```
void glutSolidTorus(GLdouble innerRadius,
GLdouble outerRadius,GLint nsides,GLint rings);
void glutWireTorus(GLdouble innerRadius,
GLdouble outerRadius,GLint nsides,GLint rings);
```

innerRadius : نصف القطر الداخلي للطارة .

outerRadius : نصف القطر الخارجي للطارة .

nsides : عدد جوانب كل مقطع دائري.

rings : عدد التقسيمات الدائرية للطارة.

5. glutSolidDodecahedron,glutWireDodecahedron

يرسم هذان الأمران العنصر الهندسي *Dodecahedron* (عنصر هندسي بـ *12* جانب منتظم) بشكل مصمت وسلكي على التوالي.

الشكل العام لهذين الأمرين:

```
void glutSolidDodecahedron(void);
void glutWireDodecahedron(void);
```

6. glutSolidOctahedron,glutWireOctahedron

يرسم هذان الأمران العنصر الهندسي *Octahedron* (عنصر هندسي بـ *8* جوانب منتظمة) بشكل مصمت وسلكي على التوالي.

الشكل العام لهذين الأمرين:

```
void glutSolidOctahedron(void);
void glutWireOctahedron(void);
```

7. glutSolidTetrahedron,glutWireTetrahedron

يرسم هذان الأمران العنصر الهندسي *Tetrahedron* (عنصر هندسي بـ *4* جوانب منتظمة) بشكل مصمت وسلكي على التوالي.

الشكل العام لهذين الأمرين:

```
void glutSolidTetrahedron(void);
void glutWireTetrahedron(void);
```

8. glutSolidIcosahedron,glutWireIcosahedron

يرسم هذان الأمران العنصر الهندسي *Icosahedron* (عنصر هندسي بـــ *20* جانب منتظم) بشكل مصمت وسلكي على التوالي.

الشكل العام لهذين الأمرين:

```
void glutSolidIcosahedron(void);
void glutWireIcosahedron(void);
```

9. glutSolidTeapot,glutWireTeapot

يرسم هذان الأمران إبريق شاي مصمت و سلكي على التوالي.

الشكل العام لهذين الأمرين:

```
void glutSolidTeapot(GLdouble size);
void glutWireTeapot(GLdouble size);
```

size :الحجم النسبي (نصف القطر) لإبريق الشاي.

الملحق د

المكتبة Auxiliary Library

يشرح هذا الملحق أشهر أوامر هذه المكتبة ، فهي تحوي أوامر لرسم عناصر هندسية ثلاثية البعد مصمتة أو سلكية .

وهذه الأوامر هي :

١ـ كرة ***Sphere***

٢ـ مكعب ***Cube***

٣ـ صندوق متوازي مستطيلات ***Box***

٤ـ طارة ***Torus***

٥ـ أسطوانة ***Cylinder***

٦ـ العنصر الهندسي ***Icosahedron***

٧ـ العنصر الهندسي ***Octahedron***

٨ـ العنصر الهندسي ***Tetrahedron***

٩ـ العنصر الهندسي ***Dodecahedron***

١٠ـ مخروط ***Cone***

١١ـ إبريق شاي ***TeaPot***

١. كرة Sphere

الشكل العام لأمر إنشاء كرة:

```
voidauxWireSphere(Gldouble radius);
void auxSolidSphere(GLdouble radius);
```

يستخدم الأمر الأول لرسم كرة سلكية أما الأمر الثاني فيستخدم لرسم كرة مصمتة .

نحتاج لرسم كرة إلى تحديد نصف قطرها *radius* .



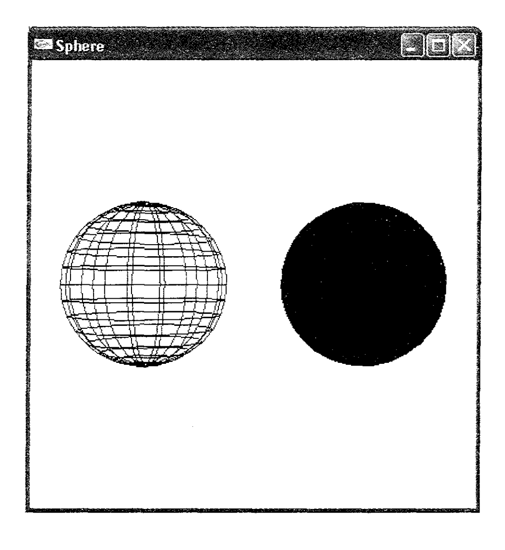
أمثلة

- رسم كرة سلكية نصف قطرها ***radius=15***

```
auxWireSphere(15);
```

- رسم كرة مصمتة نصف قطرها ***radius=15***

```
auxSolidSphere(15);
```



٢. مكعب Cube

الشكل العام لأمر إنشاء مكعب:

```
voidauxWireCube(Gldouble size);
void auxSolidCube(GLdouble size);
```

يستخدم الأمر الأول لرسم مكعب سلكي أما الأمر الثاني فيستخدم لرسم مكعب مصمت . نحتاج لرسم مكعب إلى تحديد طول الضلع *size* (أطوال أضلاع الكعب متساوية) .



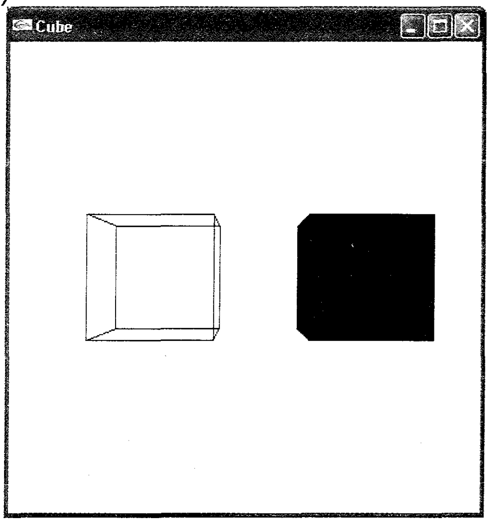
أمثلة

- رسم مكعب سلكي طول ضلعه ***size=20***

```
auxWireCube(20);
```

- رسم مكعب مصمت طول ضلعه ***size=20***

```
auxSolidCube(20);
```



٣. صندوق متوازي مستطيلات *Box*

الشكل العام لأمر إنشاء صندوق:

```
void auxWireBox(GLdouble width, GLdouble height, GLdouble depth);
void auxSolidBox(GLdouble width, GLdouble height, GLdouble depth);
```

يستخدم الأمر الأول لرسم صندوق سلكي أما الأمر الثاني فيستخدم لرسم صندوق مصمت . نحتاج لرسم صندوق إلى تحديد العرض *width* والارتفاع *height* والعمق *depth*.

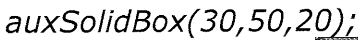


أمثلة

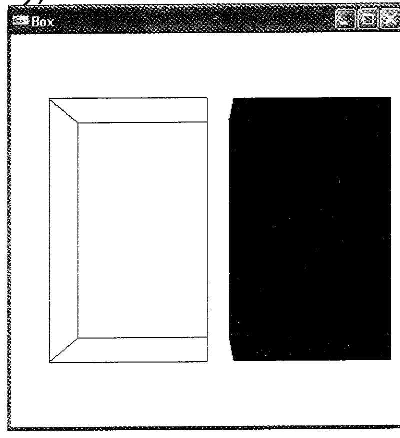
- رسم صندوق سلكي عرضه ***width=30*** وارتفاعه ***height=50*** وعمقه ***depth=20***

```
auxWireBox(30,50,20);
```

- رسم صندوق مصمت عرضه ***width=30*** وارتفاعه ***height=50*** وعمقه ***depth=20***



```
auxSolidBox(30,50,20);
```



٤. طارة Torus

الشكل العام لأمر إنشاء طارة:

```
void auxWireTorus(GLdouble innerRadius, GLdouble
outerRadius);
void auxSolidTorus(GLdouble innerRadius, GLdouble outerRadius);
```

يستخدم الأمر الأول لرسم طارة سلكية أما الأمر الثاني فيستخدم لرسم طارة مصمتة .

نحتاج لرسم طارة إلى تحديد نصف قطر الطارة (نصف قطر سماكة الطارة) *innerRadius* ونصف القطر الخارجي (من المركز حتى منتصف سماكة الطارة) *outerRadius* .



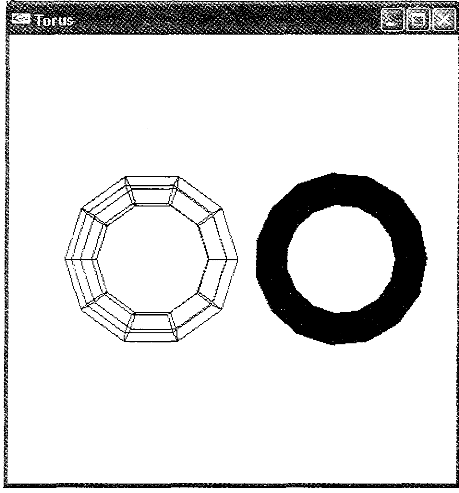
أمثلة

- **رسم طارة سلكية نصف قطر سماكتها *innerRadius=3* ونصف قطرها الخارجي *outerRadius=13***

```
auxWireTorus(3,13);
```

- **رسم طارة مصمتة نصف قطر سماكتها *innerRadius=3* ونصف قطرها الخارجي *outerRadius=13***

```
auxSolidTorus(3,13);
```



٥. أسطوانة Cylinder

الشكل العام لأمر إنشاء أسطوانة:

```
void auxWireCylinder(GLdouble radius, GLdouble height);
void auxSolidCylinder(GLdouble radius, GLdouble height);
```

يستخدم الأمر الأول لرسم أسطوانة سلكية أما الأمر الثاني فيستخدم لرسم أسطوانة مصمتة . نحتاج لرسم أسطوانة إلى تحديد نصف قطر قاعدتها *radius* وارتفاعها *height*.

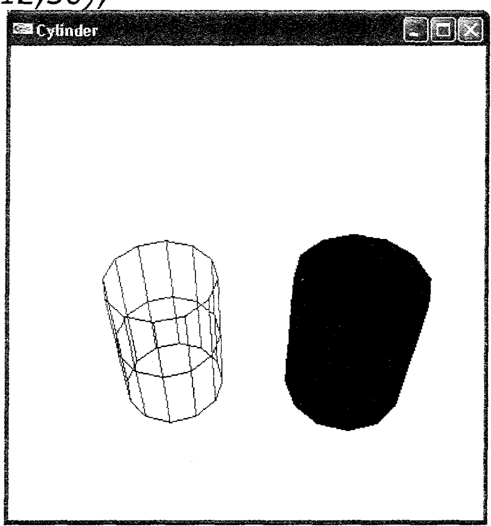

أمثلة

- **رسم أسطوانة سلكية نصف قطر قاعدتها *radius=10* وارتفاعها *height=30***

```
auxWireCylinder(10,30);
```

- **رسم أسطوانة سلكية نصف قطر قاعدتها *radius=12* وارتفاعها *height=30***

```
auxSolidCylinder(12,30);
```



٦. العنصر الهندسي *Icosahedron*

الشكل العام لأمر إنشاء **Icosahedron** (عنصر هندسي يعتمد *12* وجه في إنشائه):

```
void auxWireIcosahedron(GLdouble radius);
void auxSolidIcosahedron(GLdouble radius);
```

يستخدم الأمر الأول لرسم *Icosahedron* سلكي أما الأمر الثاني فيستخدم لرسم *Icosahedron* مصمت . نحتاج لرسم *Icosahedron* إلى تحديد نصف قطرها *radius.*



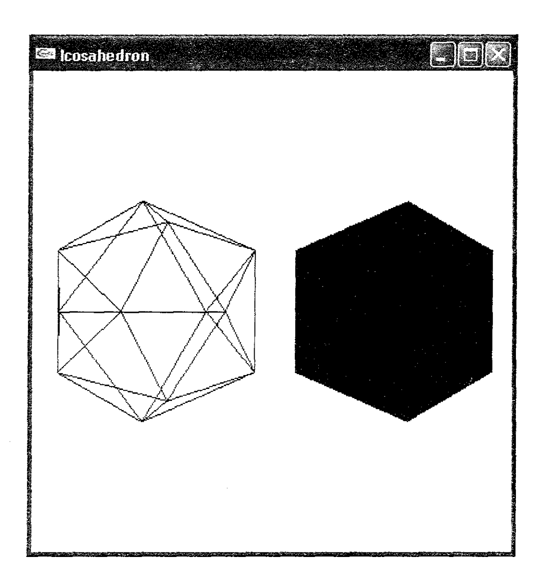
أمثلة

- رسم ***Icosahedron***سلكي نصف قطره ***radius=20***

```
auxWireIcosahedron(20);
```

- رسم ***Icosahedron***مصمت نصف قطره ***radius=20***

```
auxSolidIcosahedron(20);
```



٧. العنصر الهندسي *Octahedron*

الشكل العام لأمر إنشاء *Octahedron* (عنصر هندسي يعتمد ثمانية أوجه في إنشائه):

```
void auxWireOctahedron(GLdouble radius);
void auxSolidOctahedron(GLdouble radius);
```

يستخدم الأمر الأول لرسم *Octahedron* سلكي أما الأمر الثاني فيستخدم لرسم *Octahedron* مصمت . نحتاج لرسم *Octahedron* إلى تحديد نصف القطر *radius* .



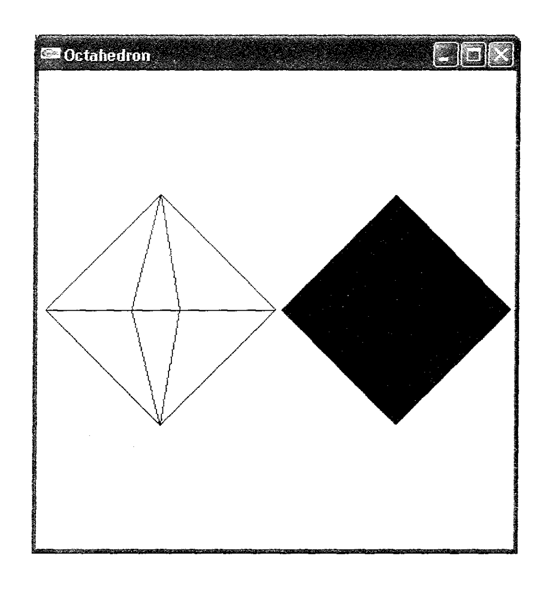
أمثلة

- رسم ***Octahedron*** **سلكي نصف قطره** ***radius=20***

```
auxWireOctahedron(20);
```

- رسم ***Octahedron*** **مصمت نصف قطره** ***radius=20***

```
auxSolidOctahedron(20);
```



٨. العنصر الهندسي *Tetrahedron*

الشكل العام لأمر إنشاء *Tetrahedron*(عنصر هندسي يعتمد أربعة أوجه في إنشائه):

```
void auxWireTetrahedron(GLdouble radius);
void auxSolidTetrahedron(GLdouble radius);
```

يستخدم الأمر الأول لرسم *Tetrahedron*سلكي أما الأمر الثاني فيستخدم لرسم *Tetrahedron*مصمت . نحتاج لرسم *Tetrahedron*إلى تحديد نصف قطره *radius* .



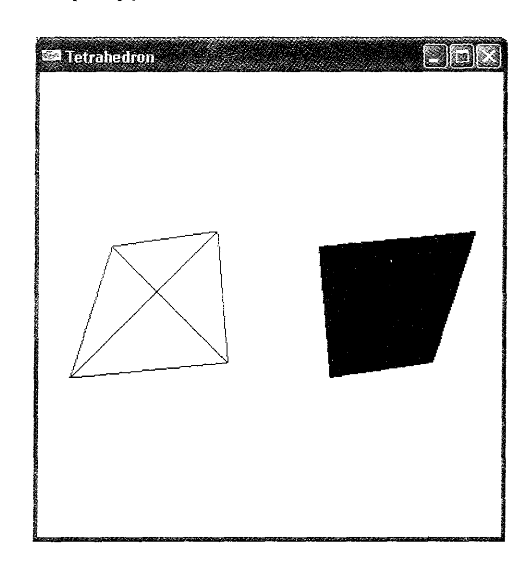
أمثلة

- رسم ***Tetrahedron*سلكي نصف قطره *radius=20***

```
auxWireTetrahedron(20);
```

- رسم ***Tetrahedron*مصمت نصف قطره *radius=20***

```
auxSolidTetrahedron(20);
```



9. العنصر الهندسي *Dodecahedron*

الشكل العام لأمر إنشاء *Dodecahedron* (عنصر هندسي يعتمد 20 وجه في إنشائه):

```
void auxWireDodecahedron(GLdouble radius);
void auxSolidDodecahedron(GLdouble radius);
```

يستخدم الأمر الأول لرسم *Dodecahedron* سلكي أما الأمر الثاني فيستخدم لرسم *Dodecahedron* مصمت . نحتاج لرسم *Dodecahedron* إلى تحديد نصف قطره *radius* .

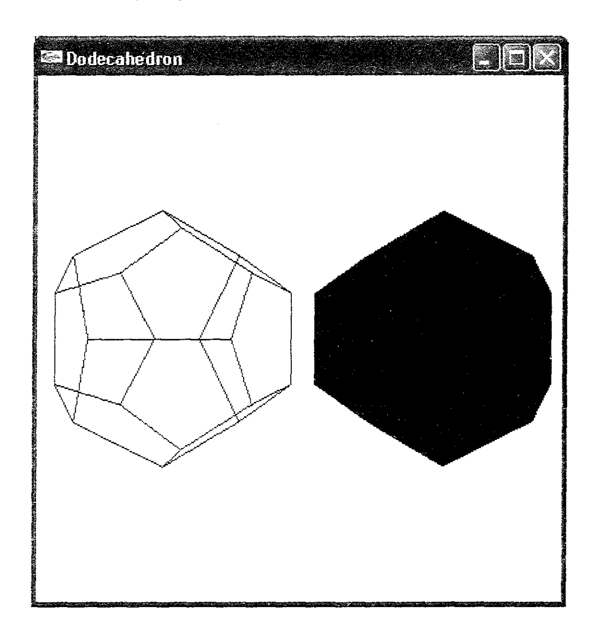

أمثلة

- **رسم *Dodecahedron* سلكي نصف قطره *radius=20***

```
auxWireDodecahedron(20);
```

- **رسم *Dodecahedron* مصمت نصف قطره *radius=20***

```
auxSolidDodecahedron(20);
```



١٠. مخروط Cone

الشكل العام لأمر إنشاء مخروط:

```
void auxWireCone(GLdouble radius, GLdouble height);
void auxSolidCone(GLdouble radius, GLdouble height);
```

يستخدم الأمر الأول لرسم مخروط سلكي أما الأمر الثاني فيستخدم لرسم مخروط مصمت . نحتاج لرسم مخروط إلى تحديد نصف قطر القاعدة *radius* والارتفاع *height* .



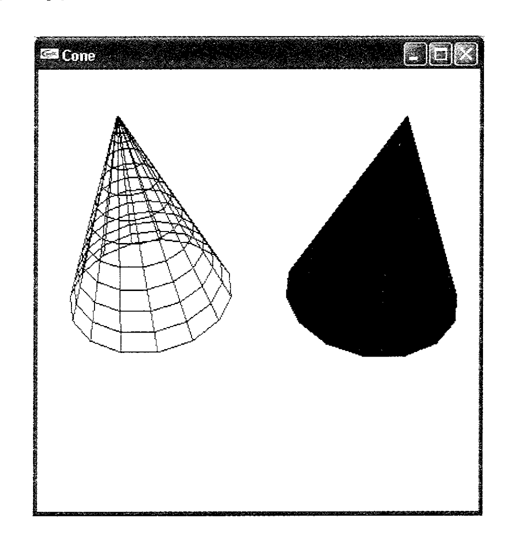
أمثلة

- رسم مخروط سلكي نصف قطر قاعدته ***radius=15*** وارتفاعه ***height=35***

```
auxWireCone(15,35);
```

- رسم مخروط مصمت نصف قطر قاعدته ***radius=16*** وارتفاعه ***height=35***

```
auxSolidCone(16,35);
```



١١. إبريق شاي ***TeaPot***

الشكل العام لأمر إنشاء إبريق شاي:

```
void auxWireTeapot(GLdouble size);
void auxSolidTeapot(GLdouble size);
```

يستخدم الأمر الأول لرسم إبريق شاي سلكي أما الأمر الثاني فيستخدم لرسم إبريق شاي مصمت . نحتاج لرسم إبريق شاي إلى تحديد نصف قطر القاعدة *size* .

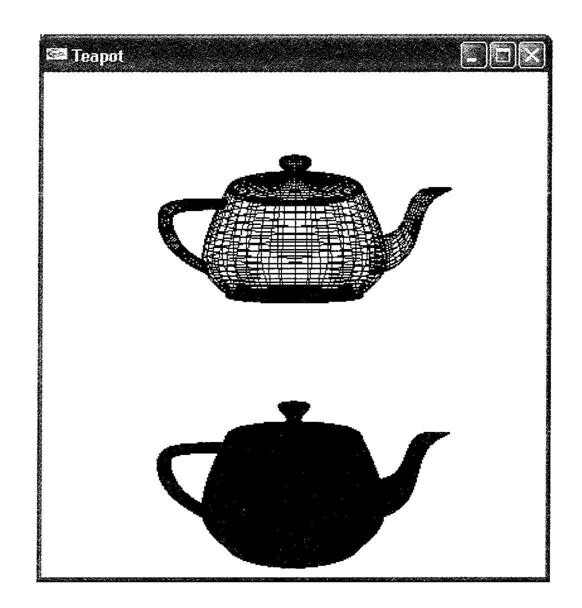

أمثلة

- **رسم إبريق شاي سلكي نصف قطر قاعدته *size=15***

```
auxWireTeapot(15);
```

- **رسم إبريق شاي مصمت نصف قطر قاعدته *size=15***

```
auxSolidTeapot(15);
```



المراجع المستخدمة

الكتب

OpenGL Programming Guide (The Red Book) .
OpenGL Reference Manual .

مواقع الانترنت

www.opengl.org
www.gamedev.net
www.mevis.de/~uwe/opengl/opengl.html



جدول المحتويات

المقدمة.. ٥

الفصل ١ مقدمة إلى OpenGL.. ٩

ما هي OpenGL ١٠

لماذا OpenGL ١١

من يتحكم بـ OpenGL ١١

كيف تعمل OpenGL ١٢

صيغة أوامر OpenGL ١٣

OpenGL كآلة حالة ١٤

المكتبات المرتبطة بـ OpenGL ١٥

برنامج إطار OpenGL ٢٠

تطبيقات عملية ٢٣

الفصل ٢ رسم العناصر الهندسية ٢٩

المحاور و المستويات الإحداثية ٣٠

مسح إطار OpenGL إلى لون معين ٣٠

النقاط (الرؤوس) و الخطوط و المضلعات ٣٤

أساسيات الرسم الهندسي في OpenGL.. ٣٨

الشكل العام للأمرين glBegin( ) و glEnd( ).. ٣٨

إظهار النقاط و الخطوط و المضلعات ٤٢

النواظم (الأشعة الاعتيادية) Normal Vectors.. ٤٦

تطبيقات عملية ٤٧

الفصل ٣ الرؤية و الإظهار ٥٩

مبدأ عمل الكاميرا ٦٠

تحويلات modeling.. ٦٤

تحويلات viewing ٦٧
تحويل الإسقاط projection ٦٩
تحويل viwport ٧١
سطوح قطع إضافية ٧٣
تطبيقات عملية ٧٤

الفصل ٤ لوائح الإظهار ٨٣

مفهوم لائحة الإظهار ٨٤
إنشاء لائحة إظهار ٨٥
التعامل مع لوائح الإظهار ٨٧
تطبيقات عملية ٨٨

الفصل ٥ الألوان ٩٥

تمييز الألوان ٩٦
ألوان الحاسب ٩٧
نمط RGBA ونمط Color-index ٩٨
تخصيص نمط تظليل ١٠٣
تطبيق عملي ١٠٤

الفصل ٦ الإضاءة ١٠٧

مفهوم الإضاءة ١٠٨
ألوان المواد ١٠٩
خطوات إضافة إضاءة إلى المشهد ١٠٩
تطبيقات عملية ١١٠

الفصل ٧ المزج و الصقل و الضباب ١٢٥

مفهوم المزج ١٢٦
المزج الثلاثي الأبعاد مع الذاكرة المؤقتة للعمق ١٣٠
الصقل ١٣٤

الضباب ١٤١
تطبيقات عملية ١٤٣

الفصل ٨ رسم النقاط الضوئية و الصور النقطية ١٤٩

الصور النقطية و الخطوط Bitmaps and Fonts ١٥٠
الصور Images ١٥٦
تطبيقات عملية ١٦١

الفصل ٩ إكساء العناصر الهندسية ١٦٧

مقدمة ١٦٨
خطوات إكساء عنصر بتركيب Texture ١٦٨
إسناد إحداثيات التراكيب ١٧١
تكرار وحصر التراكيب Textures ١٧٢
تطبيقات عملية ١٧٥

الملحق آ المكتبة GL ١٩٦

الملحق ب المكتبة GLU ٢٢٣

الملحق ج المكتبة GLUT ٢٣٧

الملحق د المكتبة Auxiliary Library ٢٥٥

جدول المحتويات ٢٦٩